

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**OPTIMAL PARACHUTE GUIDANCE, NAVIGATION, AND
CONTROL FOR THE
AFFORDABLE GUIDED AIRDROP SYSTEM (AGAS)**

by

Timothy Alphonzo Williams

June 2000

Thesis Co-Advisors:

Isaac. I. Kaminer
Oleg A. Yakimenko

Approved for public release; distribution is unlimited.

ERIC QUALITY INSPECTED 4

20000807 057

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2000		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Optimal Parachute Guidance, Navigation, and Control for the Affordable Guided Airdrop System (AGAS)			5. FUNDING NUMBERS	
6. AUTHOR(S) Williams, Timothy Alphonzo				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This study is a continuation of a previous work concerning the Affordable Guided Airdrop System (AGAS), a parachute system that integrates low-cost guidance and control into fielded cargo air delivery systems. This thesis sought to expand upon the previous study and provide more information and research on this innovative and critical military system. Several objectives and tasks were completed in the course of this research and development. The simulation model used in the previous work for feasibility and analysis studies was moved from a MATLAB/SIMULINK® environment to a MATRIX-X® environment in anticipation of AGAS future use on an Integrated Systems, Incorporated AC-104 real-time controller. Further simulation and study for this thesis were performed on the new system. The new model implemented characteristics of the G-12 parachute, which eventually will be used in the actual flight testing of the AGAS airdrop. The system of pneumatic muscle actuators (PMAs) built by Vertigo, Incorporated and used on the AGAS was modeled on the computer also. The characteristics of this system and their effects on AGAS guidance and control were studied in depth. The control concept of following a predicted trajectory based on certain wind predictions and other ideas for control algorithms to minimize fuel gas usage, number of control actuations and final control error were also studied. Conclusions and recommendations for further study were drawn from this project.				
14. SUBJECT TERMS MATRIX-X® Software, Parachute, Guidance, Navigation, Control, Simulation, Wind Estimation			15. NUMBER OF PAGES 134	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**OPTIMAL PARACHUTE GUIDANCE, NAVIGATION, AND CONTROL FOR
THE AFFORDABLE GUIDED AIRDROP SYSTEM (AGAS)**

Timothy Alphonzo Williams
Ensign, United States Navy
B.S., United States Naval Academy, 1999

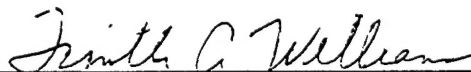
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

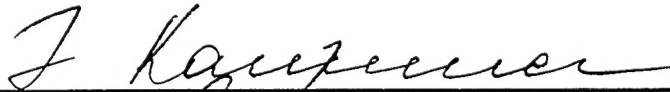
**NAVAL POSTGRADUATE SCHOOL
June 2000**

Author:

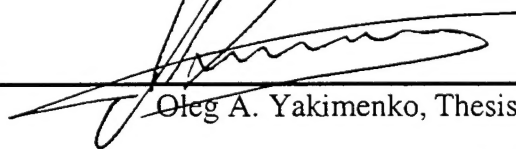


Timothy Alphonzo Williams

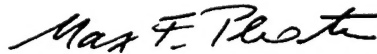
Approved by:



Isaac I. Kaminer, Thesis Co-Advisor



Oleg A. Yakimenko, Thesis Co-Advisor



Maximilian F. Platzer, Chairman
Department of Aeronautics and Astronautics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study is a continuation of a previous work concerning the Affordable Guided Airdrop System (AGAS), a parachute system that integrates low-cost guidance and control into fielded cargo air delivery systems. This thesis sought to expand upon the previous study and provide more information and research on this innovative and critical military system. Several objectives and tasks were completed in the course of this research and development. The simulation model used in the previous work for feasibility and analysis studies was moved from a MATLAB/SIMULINK[®] environment to a MATRIX-X[®] environment in anticipation of AGAS future use on an Integrated Systems, Incorporated AC-104 real-time controller. Further simulation and study for this thesis were performed on the new system. The new model implemented characteristics of the G-12 parachute, which eventually will be used in the actual flight testing of the AGAS airdrop. The system of pneumatic muscle actuators (PMAs) built by Vertigo, Incorporated and used on the AGAS was modeled on the computer also. The characteristics of this system and their effects on AGAS guidance and control were studied in depth. The control concept of following a predicted trajectory based on certain wind predictions and other ideas for control algorithms to minimize fuel gas usage, number of control actuations and final control error were also studied. Conclusions and recommendations for further study were drawn from this project.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION	1
II. MODELING DESCRIPTION.....	3
A. PARACHUTE DYNAMICS MODELING.....	4
B. GPS/HEADING SENSOR MODELING.....	13
1. Global Positioning System (GPS).....	13
2. Heading Sensor/Compass	19
C. CONTROL SYSTEM MODELING.....	21
1. Control Strategies	21
2. Parachute Control Logic	23
D. ACTUATOR SYSTEM MODELING	34
III. DATA COLLECTION.....	51
A. ACTUATORS.....	51
B. WINDS.....	58
IV. SIMULATION.....	63
A. PROCESS	63
B. RESULTS	63
V. CONCLUSIONS AND RECOMMENDATIONS	77
A. CONCLUSIONS.....	77
B. RECOMMENDATIONS.....	77
LIST OF REFERENCES.....	81
APPENDIX A. SCRIPT FILES USED IN SIMULATION	83
APPENDIX B. ADDITIONAL FIGURE AND CODE	105
INITIAL DISTRIBUTION LIST	107

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Dynamics Model From Ref. [8].....	4
Figure 2. Vehicle Model.....	7
Figure 3. Aerodynamics Model.....	7
Figure 4. Mass Properties Model.....	9
Figure 5. Selective Availability on Error Model.....	14
Figure 6. SystemBuild® Selective Availability Errors.....	15
Figure 7. MATLAB® Selective Availability Errors.....	16
Figure 8. Selective Availability True Position vs. GPS Position.....	17
Figure 9. GPS SA off Model.....	17
Figure 10. GPS Jitter Model.....	18
Figure 11. SystemBuild® GPS Errors.....	18
Figure 12. MATLAB® GPS Errors.....	19
Figure 13. Heading Sensor Model.....	20
Figure 14. XMATH® Modeled Heading Sensor Error.....	20
Figure 15. MATLAB® Modeled Heading Sensor Error.....	21
Figure 16. Flow Diagram for Parachute Control Logic.....	24
Figure 17. Operating Angles.....	25
Figure 18. Tolerance Cone Guidance Concept From Ref. [12].....	27
Figure 19. Tolerance Cone used in Simulation.....	29
Figure 20. Tolerance Cone State Diagram.....	30
Figure 21. Controller.....	32
Figure 22. Control Logic with Good Wind Prediction.....	33
Figure 23. Vertigo, Inc. Actuator System Concept From Ref. [13].....	35
Figure 24. Actuator Modeling Concept.....	36
Figure 25. PMA Model.....	42
Figure 26. Force Rotation with Length Change.....	43
Figure 27. Number of Controls and Corresponding Glide Ratio.....	44
Figure 28. Counter.....	45
Figure 29. Counter with Thresholds.....	47
Figure 30. Actuator Data from Simulation.....	48
Figure 31. Control Logic for Actuator Data Simulation.....	49
Figure 32. Four Pneumatic Muscle Actuators (PMAs) From Ref. [8].....	51
Figure 33. Change in Length of Riser vs. PMA Set Pressure.....	52
Figure 34. Increasing Fill Time vs. Reservoir Pressure (Vertigo Test) From Ref. [14]..	53
Figure 35. Fill Time vs. Tank Pressure (YPG Test).....	54
Figure 36. Fill Time and Tank Pressure vs. Fill Cycle Number.....	55
Figure 37. Change in Tank Pressure vs. Fill Number.....	57
Figure 38. Magnitude of Measured Wind.....	59
Figure 39. Measured Wind Direction.....	59

Figure 40. Non-Controlled Parachutes Subjected to Hourly Winds	60
Figure 41. Trajectory-Seeking Control Strategy Age-of-Wind Comparison	64
Figure 42. Target-Seeking Control Strategy Age-of-Wind Comparison	64
Figure 43. Distribution of Ages of Wind Predictions (437 samples).....	67
Figure 44. Trajectory-Seek Impact and Release Points	68
Figure 45. Target-Seek Impact and Release Points	68
Figure 46. Trajectory-Seek Impact Points Zoomed In.....	69
Figure 47. Target-Seek Impact Points Zoomed In.....	70
Figure 48. Distribution of Parachute Landing Miss Distances.....	70
Figure 49. Thirty Trajectories for Trajectory-Seek and Target-Seek Strategies	71
Figure 50. Fill Time vs. Tank Pressure for Five Actuator Models	73
Figure 51. Actuator-in-the-Loop Testing Scheme	79

LIST OF TABLES

Table 1. Table of Deflation Times	56
Table 2. Control Errors for Simulation	65
Table 3. Statistical Data on Control Strategy Comparison.....	72
Table 4. Actuator Characteristic Research Simulation Results	74

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

The author would like to thank many people without whom this project would not have been completed. Thanks go to Scott Dellicker and everyone at Yuma Proving Grounds and the U.S. Army Soldier and Biological Chemical Command who provided an abundance of information, guidance and crucial financial support. To Richard Benney, Phil Hattis, Bob Wright and the many others who have been working diligently since the project began. To Professors Isaac Kaminer, Richard Howard, and Oleg Yakimenko for providing the necessary tools and insight for my research. To LT Charles Hewgley for his help in the research, analysis, and modeling; much credit goes to you. And thanks to the close friends and family who provided humor and motivation over the long haul. I am greatly indebted to you all.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The Affordable Guided Airdrop System (AGAS) has progressed significantly since its inception following the United States Air Force Advisory Board's call for improvements in the area of supply airdrop in 1997 [Ref. 1]. The team of engineers and scientists from the US Army Soldier and Biological Chemical Command, Yuma Proving Grounds, Draper Laboratories, Planning Systems, Inc., Vertigo, Inc., Parks College Parachute Research Group, the Naval Postgraduate School, and several other research groups have contributed greatly to making this endeavor a very promising undertaking.

The problems involved in supply airdrop can be traced back to World War II. Airdrop by the Allies on the Western Front was often a guessing game, and with swirling winds and bad weather, many times the food and equipment would fall into the wrong hands. The Marines in the Chosin Reservoir during the Korean War lost practically all of their supply reinforcements to the Chinese because of inaccurate airdrop. Recent developments in the Persian Gulf, Haiti, Somalia, and Bosnia, and humanitarian efforts to third-world countries have experienced the same frustration.

The main impetus behind the development of the AGAS system is affordability. Large-scale parafoil systems have already been developed as predecessors to this system. These systems have proven successful. However, the AGAS system would take advantage of supplies already available to the military in the creation of an affordable, yet smart and reliable system.

The structural modeling of a parachute's dynamics is a very difficult process. The parachute's aerodynamics are governed by extremely complex equations [Ref. 2]. The difficulty in the guidance of a parachute and the modeling of this phenomenon are equally as difficult. Add in the variability of the winds, of which a parachute is much more influenced than a rigid aircraft, and one has quite a tricky problem.

Much work and analysis have gone into the aerodynamic modeling of parachute dynamics. Most recently this problem has been studied by White and Wolf [Ref. 3], Tory and Ayres [Ref. 4], and Doherr and Salarias [Ref. 5]. For the AGAS, this problem is being researched by engineers at Draper Laboratories, Parks College Parachute Research Group, and the Naval Postgraduate School.

The variability of winds and wind prediction are being studied by engineers at Planning Systems, Incorporated. The ultimate goal of this analysis is to provide an on-board weather communications and data processing system enabling the AGAS to deliver the parachute drops from very high altitudes in all weather and terrain [Ref. 6]. The variability of winds around even flat terrains where such variability is not expected has proven to make this analysis more difficult than anticipated, but a crucial piece to the puzzle.

The guidance system is a collective effort of students, scientists, and engineers at the Naval Postgraduate School and Vertigo, Incorporated. Vertigo is providing design and analysis of the actuators used on the system. The Naval Postgraduate School is studying the optimization and testing of the parachute system, of which the body of this thesis is mostly devoted.

II. MODELING DESCRIPTION

The initial objective of this project was to move the computer model of the C-9 parachute's dynamics, sensor package, and control system completed by Scott Dellicker on MATLAB/SIMULINK[®] to MATRIX-X/XMATH/SystemBuild[®]. The purpose of this transition was one of anticipation. The real-time controller to be used with the AGAS system in actual testing is a guidance computer made by Integrated Systems (ISI) called the AC-104 system. This guidance computer will eventually be used to determine and activate desired control inputs based on control algorithms fed into the computer. MATRIX-X[®] provides the ability to build a control scheme through the use of script code and easy-to-use building blocks in a software package called XMATH/SystemBuild[®]. This model of the controller can then be automatically transformed into downloadable C-code through a MATRIX-X[®] program called AutoCode with very few constraints. The C-code control algorithm can then be executed by the AC-104 system.

In order to verify the working of the control algorithms on XMATH/SystemBuild[®], the entire model of the parachute dynamics, sensors, and control system had to be converted from MATLAB[®]. After this transformation, further study on the feasibility of the control system, the accuracy of the parachute dynamics, analysis of the sensor models, and other studies were done on the XMATH/SystemBuild[®] model [Ref. 7].

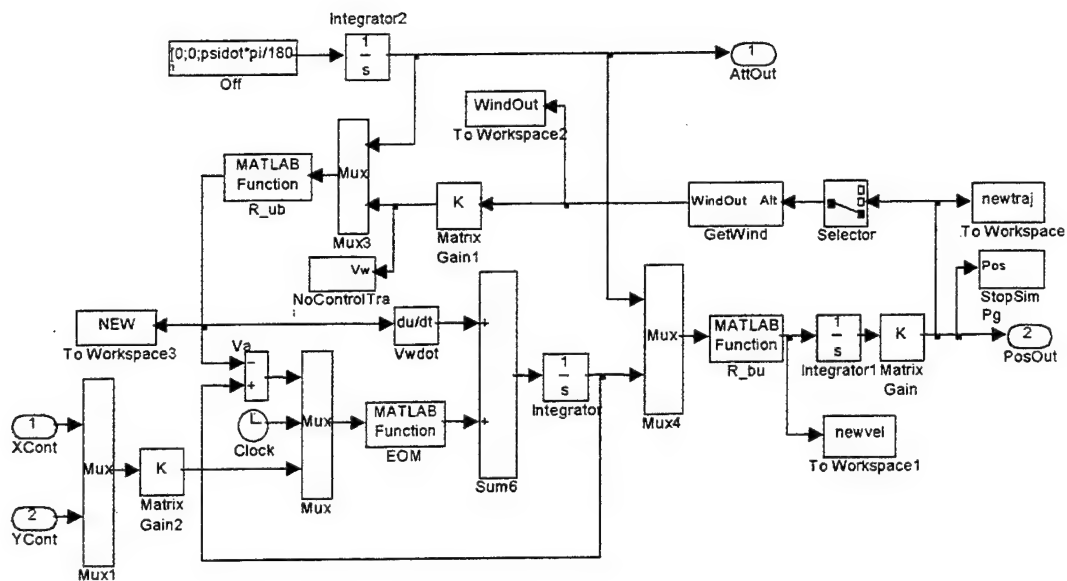


Figure 1. Dynamics Model From Ref. [8]

A. PARACHUTE DYNAMICS MODELING

Dellicker's MATLAB[®] model of the C-9 parachute system's dynamics, a subsystem aptly named "Dynamics", is shown above in Figure 1. One goal of the project was to replicate this 3-degree-of-freedom model of the parachute dynamics in XMATH[®], as well as transition from using C-9 physical parachute data (such as area and weight) to G-12 data. The discussion of this MATLAB[®] model will not be done in great detail; rather, discussion will center upon the basic premise of what the model is trying to accomplish. Thus, the implementation of the blocks and code used in XMATH/SystemBuild[®] to model the parachute's behavior will be understood with greater clarity.

The MATLAB[®] subsystem "Dynamics" describes the behavior of the parachute dynamics. It is a simplistic view of the parachute dynamics as a point-mass. The equations of motion for this very simplistic view are (in state-space form):

$$(1) \quad \dot{V}_A = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} m + \alpha_{11} & 0 & 0 \\ 0 & m + \alpha_{22} & 0 \\ 0 & 0 & m + \alpha_{33} \end{bmatrix}^{-1} \left\{ \frac{-qC_D S}{V_T} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix} + \begin{bmatrix} F_{control,x} \\ F_{control,y} \\ 0 \end{bmatrix} \right\}$$

where \dot{u} , \dot{v} , and \dot{w} are linear accelerations in the x, y, and z directions in ft/s²; u, v, and w are the corresponding airspeeds in ft/s, α_{xx} is the apparent mass in slugs, m is the mass of parachute and payload in slugs, q is the dynamic pressure in lb_f/ft², C_D is the coefficient of drag (dimensionless), S is the drag area of the parachute in ft², V_T is the magnitude of the true airspeed in ft/s, W is the weight of the payload and parachute in lb_f, and F_{control} is the force effect of the control actuators in lb_f (in only the x and y directions). This equation in its most basic form is $a = m^{-1}F$. The apparent mass terms are computed from the following equations:

$$(2) \quad \alpha_{11} = \frac{1}{4} \rho \frac{4}{3} \pi \left(\frac{D_p}{2} \right)^3;$$

$$\alpha_{22} = \alpha_{11};$$

$$\alpha_{33} = 2 \times \alpha_{11}$$

where D_p is the profile diameter of the parachute equal to 2/3 of the reference diameter of the flat circular parachute. For the G-12 the reference diameter is equal to 64 square feet.

The dynamic pressure is calculated by $q = \frac{1}{2} \rho V_T^2$. The density (ρ) changes with altitude; thus, q and the apparent mass change with altitude.

The XMath/SystemBuild[®] model of the parachute dynamics (a model called “Vehicle Model”) is shown in Figure 2. The inputs to the system are the four actuator commands (commands to turn the actuator on or off), “PMA1_cmd_psi” through “PMA4_cmd_psi”. PMA stands for Pneumatic Muscle Actuator, the braided fiber tubes that can be pressurized or depressurized in order to lengthen or shorten a parachute riser as described in Dellicker’s thesis. The PMA commands go through a block called “PMA model” (or “New PMA model”) which characterize the dynamics of the PMAs. XMath[®] blocks into which other lower level blocks can be placed to model a certain behavior are known in SystemBuild[®] as super blocks. They are similar to MATLAB[®] subsystem blocks. “PMAModel” outputs the states of the four PMAs (ranging from 0 psi for a fully vented PMA to a maximum pressure for a fully filled PMA) which then become inputs to the super block “Aerodynamics”. The PMA model will be described more fully in a later section.

The inside of the super block “Aerodynamics” is shown in Figure 3. This block describes the equations of motion for the 3-DOF parachute model (Eq. 1). This model is described as 3-degree-of-freedom because only the x , y , and z positions of the parachute are affected by control inputs. The angular positions Φ , Θ , and ψ (around the x , y , and z axis, respectively), are not affected by control in this simplified model.

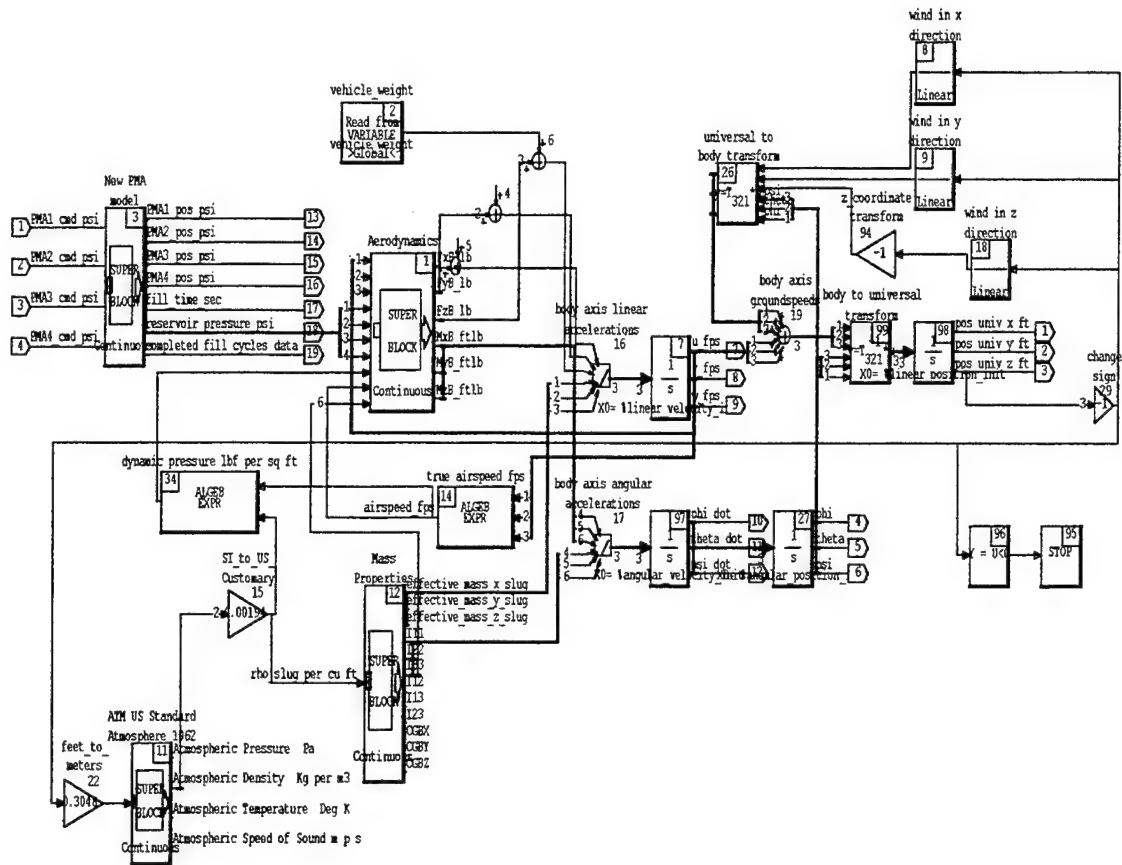


Figure 2. Vehicle Model

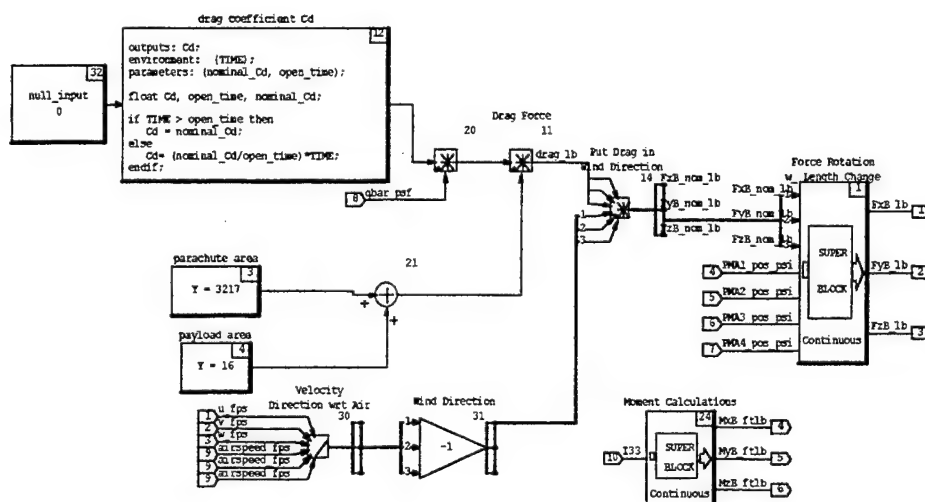


Figure 3. Aerodynamics Model

At the beginning of the simulation the parachute experiences a few seconds of free fall before the canopy opens and the parachute settles to a constant C_D . This opening period is modeled as a linear relationship between a ballistic ($0 C_D$) and the constant C_D in the drag coefficient block. The constant drag coefficient for the G-12 parachute was determined to be approximately 0.733 (from flight test). The following code describes this linear relationship and is part of the block script block "drag coefficient C_D ":

```
outputs: Cd;
environment: (TIME);
parameters: (nominal_Cd, open_time);
float Cd, open_time, nominal_Cd;
if TIME > open_time then
    Cd = nominal_Cd;
else
    Cd = (nominal_Cd/open_time)*TIME;
endif;
```

The "outputs" line in block-script declares the outputs of the code. "Environment" declares any variables that are automatically created by the simulation (in this case TIME, the simulation time). "Parameters" declares any variables used in the workspace. The next line declares the precision of each variable used in the code. The actual body of the code follows (a basic if-then statement) which sets the C_D equal to the nominal C_D (0.733) after the open time (5 seconds), or to a linear relationship between 0 and the nominal C_D for the open time of 5 seconds.

The rest of the blocks in "Aerodynamics" fully describe the equations of motion in Eq. 1. Control force is added to the equations in the super block "Force Rotation" (or "Force Rotation w/ Length Change"). In the parachute "Vehicle Model", the vehicle weight is added to the equations in only the z direction. This weight is added only in the

z direction because of the assumption that the parachute does not pitch or roll, but has a constant rotation rate. The vehicle weight is read in from a global variable called "vehicle_weight" using a read-from-variable block.

Once the forces on the parachute have been calculated by "Aerodynamics", the acceleration must be calculated by dividing the total force vector by the mass vector of the parachute and payload ($F=ma$, or in this case, $a = m^{-1}F$). Figure 4 shows the calculation of this mass vector in the super block "Mass Properties". The block calculates the apparent masses in Eq. 2. They are then added to the mass of the parachute and payload (the vehicle weight divided by $g = 32.174 \text{ ft/s}^2$) to give the effective mass in the x, y, and z directions. The forces on the parachute body calculated in "Aerodynamics" are then divided by the effective masses in each axis to give the linear accelerations in each axis.

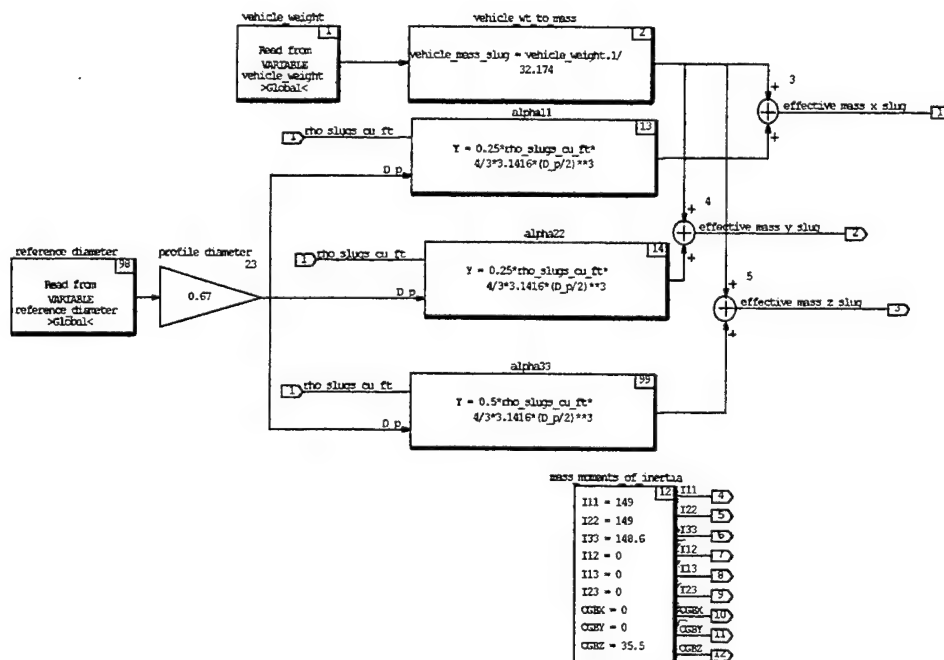


Figure 4. Mass Properties Model

The "Mass Moments of Inertia" block gives the values of the inertia matrix *if* the model took into account the pitching and rolling of the parachute and displacements from the center of gravity (CGBX through CGBZ). As it stands, however, most of these values are not used, and the only value that is used is I_{33} , the mass moment of inertia about the z body-axis. A random perturbation of the angular acceleration of the body about the z axis (a perturbation from its normally assumed constant value of 0 rad/s²) is generated through a "Uniform Random Number" block in the "Moment Calculations" block in "Aerodynamics" (see Fig. 3). This random perturbation is between -1 and 1 rad/s². It is then filtered to give the rotational disturbance. The rotational disturbance is then multiplied by the mass moment of inertia, I_{33} , to give a moment disturbance about the z axis ($M_{33} = I_{33}\alpha$). The other moments, M_{11} and M_{22} , moments about the x and y axis respectively, are set to zero to support the assumption of no pitch or roll. Upon close inspection of Fig. 2, however, this moment disturbance M_{33} is divided immediately by I_{33} to give the angular acceleration disturbance once again. If the model took into account the affect of the control on the parachute moments, a moment caused by the control would be added to this moment disturbance at the point before dividing by I_{33} , but as it stands, the model is a simplistic 3-DOF representation of parachute dynamics.

The changing air density (ρ) with respect to altitude is calculated using a standard atmospheric table extracted from XMath's Aerospace Libraries[®]. The altitude in meters (converted from feet) is fed into this table and several useful air properties are interpolated from this altitude, including density. This ever-changing density is then fed

into blocks to calculate dynamic pressure (q) and apparent masses (α_{xx}) from their respective equations (converted from meters to feet).

Once the linear and angular accelerations are calculated, they are integrated to give linear and angular velocities in body-axis coordinates. Initial conditions for the parachute are specified in integrator blocks. The parachute's initial velocity in x and y reflects the initial magnitude and direction of the aircraft's velocity upon releasing the parachute. The initial speed is 130 ft/s, and the initial heading is 045 degrees for this model, but this initial velocity would change with a different aircraft velocity. The initial speed of the parachute in the downward z direction is 25 ft/s. The initial angular velocity of the parachute is 1.8 deg/sec for $\dot{\psi}$, and zero for both $\dot{\phi}$ and $\dot{\theta}$ (no change in pitch or roll angle from 0 degrees). Angular velocity is then directly integrated again to give angular positions, the Euler angles in the x , y , and z axis, with the initial conditions for all three angles being zero.

Integration of the linear accelerations actually gives the parachute's airspeed in ft/s. In order to calculate the parachute groundspeeds (in body-axis coordinates), the wind velocity must be added to the airspeeds. This wind velocity must also be transformed from universal to body-axis coordinates before being added to the body-axis airspeeds to get body-axis groundspeeds:

$$(3) \quad {}^B V_G = {}^B V_A + {}^B V_W$$

where ${}^B V_G$ is the groundspeed vector, ${}^B V_A$ is the airspeed vector, and ${}^B V_W$ is the wind speed vector, all in body-axis coordinates. The transformation matrix for converting

vectors (either position or velocity vectors) from universal (inertial) to body coordinate systems (B_R) is [Ref. 9]:

$$(4) \quad {}^B_R = \begin{bmatrix} \cos \psi \cos \theta & \sin \psi \cos \theta & -\sin \theta \\ \cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi & \sin \theta \sin \phi \sin \psi + \cos \psi \cos \phi & \cos \theta \sin \phi \\ \cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi & \sin \theta \cos \phi \sin \psi - \cos \psi \sin \phi & \cos \theta \cos \phi \end{bmatrix}$$

and the transformation from body to universal coordinates is just the inverse:

$$(5) \quad {}^U_R = {}^B_R^{-1} \text{ where } \psi, \phi, \text{ and } \theta \text{ are the parachute Euler angles.}$$

In the block "Aerodynamics", the wind velocity in universal coordinates is interpolated from different wind files. The interpolation is indexed by altitude. Since the parachute model is referenced to a north-east-down (NED) coordinate system, and the wind files are referenced to a north-east-up (NEU) coordinate system, the z position must be multiplied by a gain of -1. The interpolated winds in x, y, and z are in universal and NEU coordinates, so they must be transformed to NED body coordinates by another -1 gain block and the coordinate transformation block "Universal to Body Transform", which performs the transformation in Eq. 4. The winds can then be added to the body-axis airspeeds in order to get body-axis groundspeeds. These groundspeeds are then transformed to universal coordinates through another transformation block that performs the matrix multiplication outlined in Eq. 5.

In universal coordinates, the groundspeed can be integrated to obtain x, y, and z positions in NED coordinates. The initial position read into this integrator is the initial x offset and y offset of the drop (from the intended drop point of 0,0) and the altitude of the

drop in NED coordinates, which is always equal to -9500 ft (0 altitude is the ground point). These universal x, y, and z positions are the main outputs of the "Vehicle Model".

Finally, z position is again multiplied by a -1 gain block to be used in the wind files, the atmospheric tables, and to stop the simulation when the altitude of the parachute is zero (when it hits the ground). A logical expression and stop block perform this last task (see Fig. 2). When the statement output in the logical expression block ($Y = U \leq 0$) is TRUE, that is, when the input altitude is less than or equal to zero, the simulation is stopped. While the altitude was multiplied by -1 to perform these tasks, the output of the vehicle model is still the x, y, and z positions in NED coordinates. A diagram of the entire parachute dynamics modeling scheme is shown in Appendix B.

B. GPS/HEADING SENSOR MODELING

1. Global Positioning System (GPS)

A simple GPS model was implemented in XMATH[®] based on the MATLAB[®] model. This model simply created errors in the Cartesian (x, y, z) coordinates matching test data. Normally, errors in the ranges to the satellites as a result of several different error sources (such as receiver noise and satellite clock noise) would have to be modeled using numerical solutions such as a maximum likelihood estimation algorithm. However, these numerical solutions consume much simulation time and are therefore unsuitable for simulation on a simple personal computer, so they were not utilized. Instead, system identification tools were implemented.

Both a model with selective availability on and one with selective availability off were modeled. A variable in the workspace, "saon", determined whether selective

availability was on or off. If saon was equal to 1, then the selective availability errors were chosen and the GPS error was greater. If saon was equal to 0, selective availability was off and the GPS was very accurate.

The selective availability errors in commercial GPS receivers contain induced errors, which restrict the use of the GPS full power and precision to only authorized users such as military and other Department of Defense units. Receivers not restricted to selective availability are capable of removing the induced errors through the processing of several cryptographic codes [Ref. 10]. The AGAS is desired to utilize a commercial GPS receiver for the purposes of cost reduction. As described in Dellicker's thesis, these GPS selective availability errors were modeled using a system identification tool known as ARMAX [Ref. 8]. The re-creation of this selective availability model in XMATH[®] is shown in Fig. 5. The same discrete transfer functions and noise source blocks were used in this system. This transfer function is as follows:

$$(6) \quad T.F. = \frac{z^4 - 1.5302z^3 + 0.2608z^2 + 0.2566z + 0.0192}{z^4 - 2.6500z^3 + 1.9582z^2 + 0.0337z - 0.3420}$$

Comparisons between MATLAB[®] selective availability errors and XMATH[®]

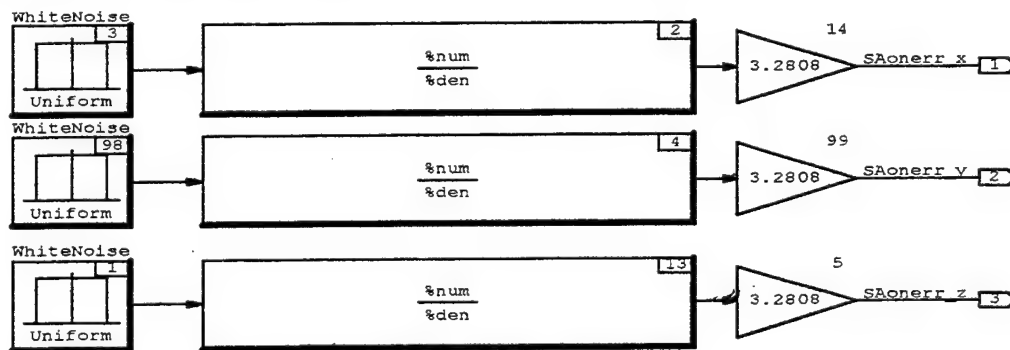


Figure 5. Selective Availability on Error Model

selective availability errors are shown in Fig. 6 and 7.

In Dellicker's thesis, selective availability errors were measured with a mean of approximately zero meters in each axis and standard deviations of 17.2, 28.8, and 21.1 m in the x, y, and z axes respectively [Ref. 8]. The old model, whose errors are sampled in Fig. 7, showed a mean of 0-2 m and standard deviations of 13-16 m. The new model

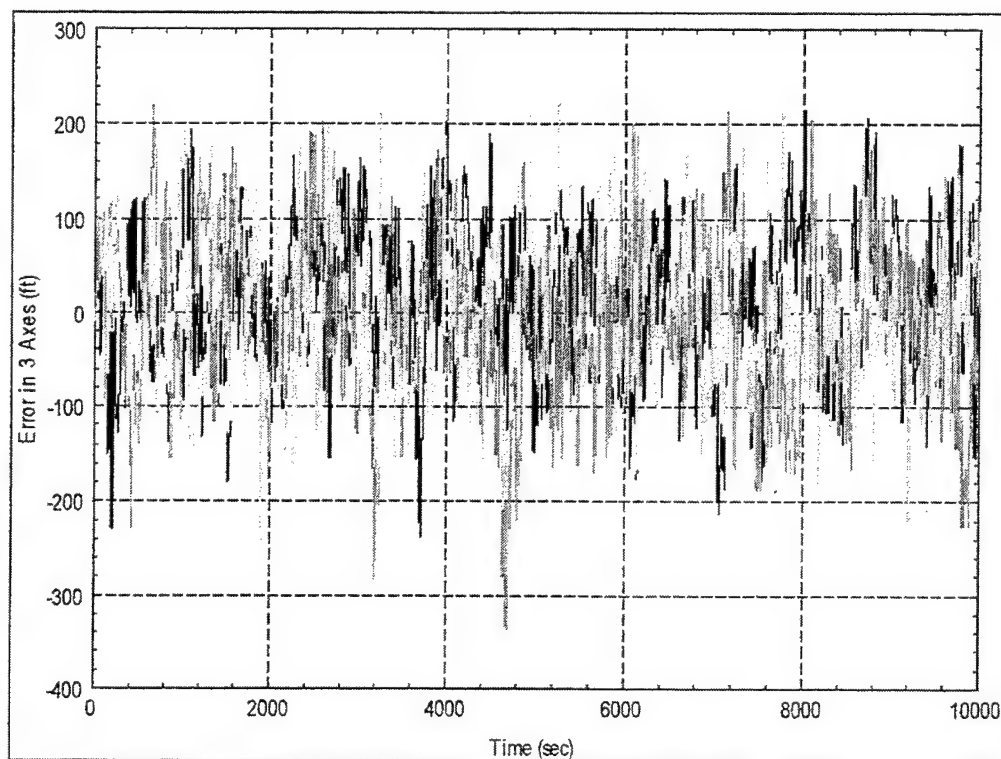


Figure 6. SystemBuild® Selective Availability Errors

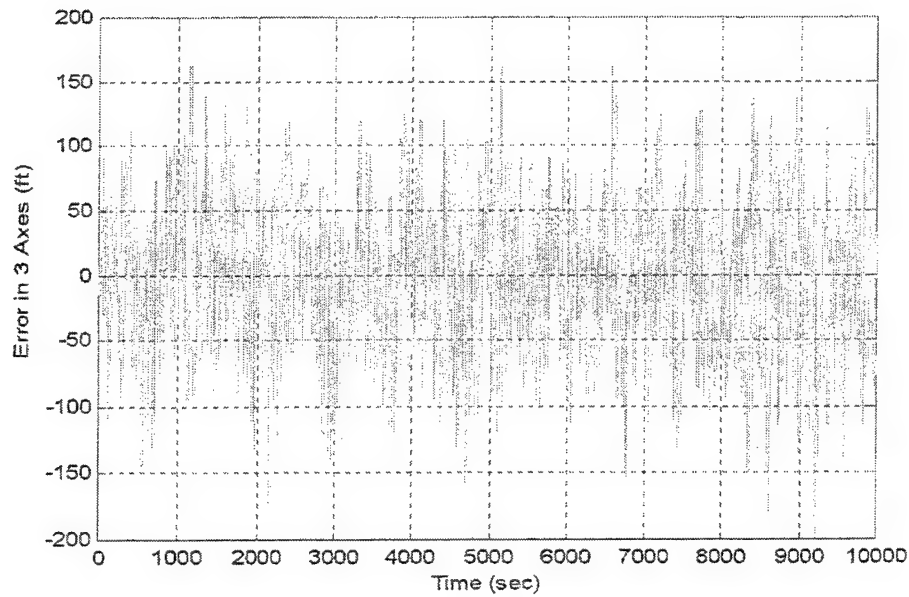


Figure 7. MATLAB[®] Selective Availability Errors

showed a mean of 0-4 m and standard deviation of 22-25 m. This new model provides an adequate picture of the selective availability errors. A sampling of errors for the new model is shown in Fig. 6.

In Fig. 8, a typical simulation run, selective availability provides an adequate estimation of position, but not quite as accurate as could be obtained with a DOD GPS receiver.

With selective availability off, GPS accuracy goes up. A comparable model to the MATLAB[®] version was implemented in XMATH[®], which included standard errors from GPS (clock error, atmospheric noise, etc.) and "jitter" errors. These two models are shown in Figures 9 and 10. Results from the MATLAB[®] simulation of the GPS errors

and the XMATH[®] simulation compared well with MATLAB[®]. These simulations modeled the errors of a Honeywell Embedded GPS/Inertial Navigation System [Ref. 8].

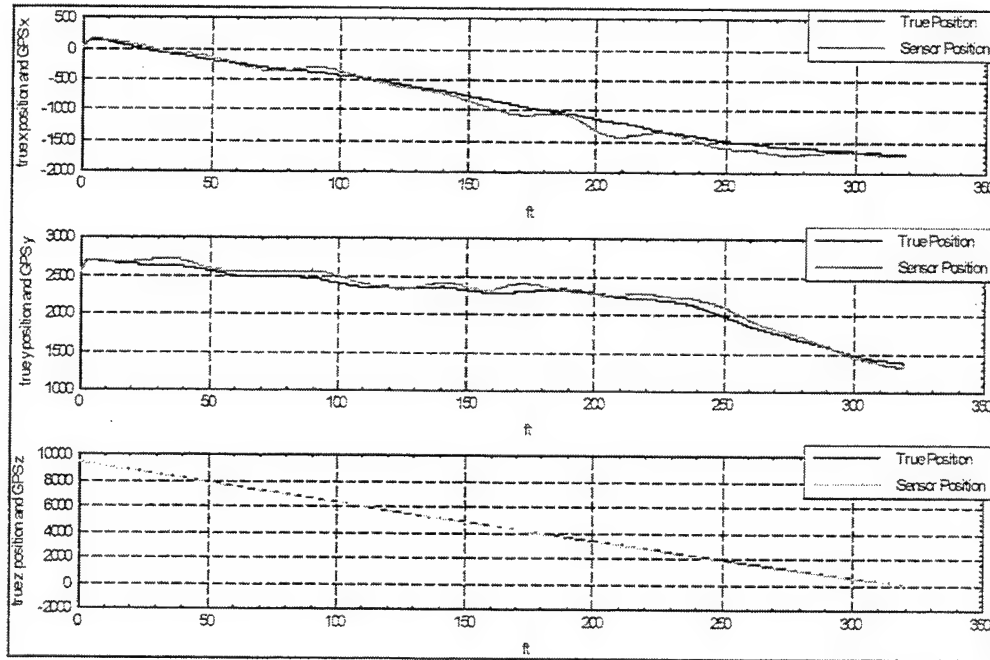


Figure 8. Selective Availability True Position vs. GPS Position

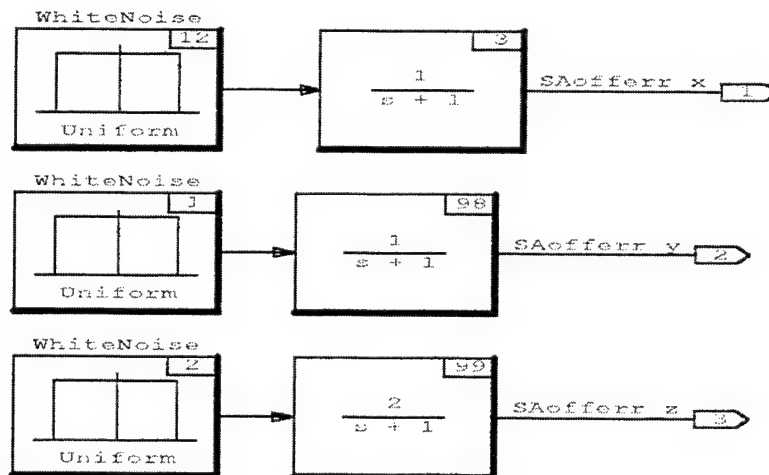


Figure 9. GPS SA off Model

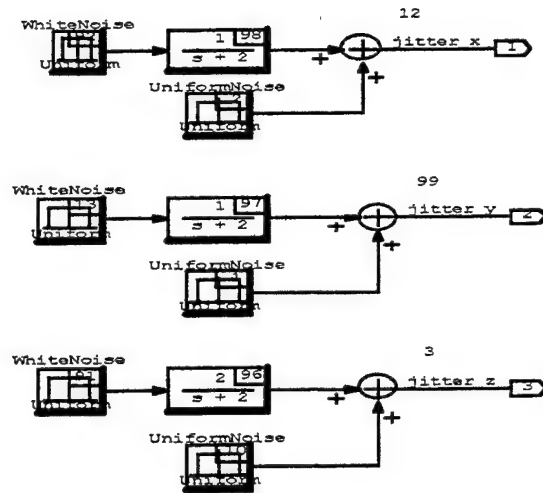


Figure 10. GPS Jitter Model

Once again, the modeled results in XMATH[®] compared well to the results from the MATLAB[®] model. The following Figures 11 and 12 show the GPS errors for both models, which are typical values.

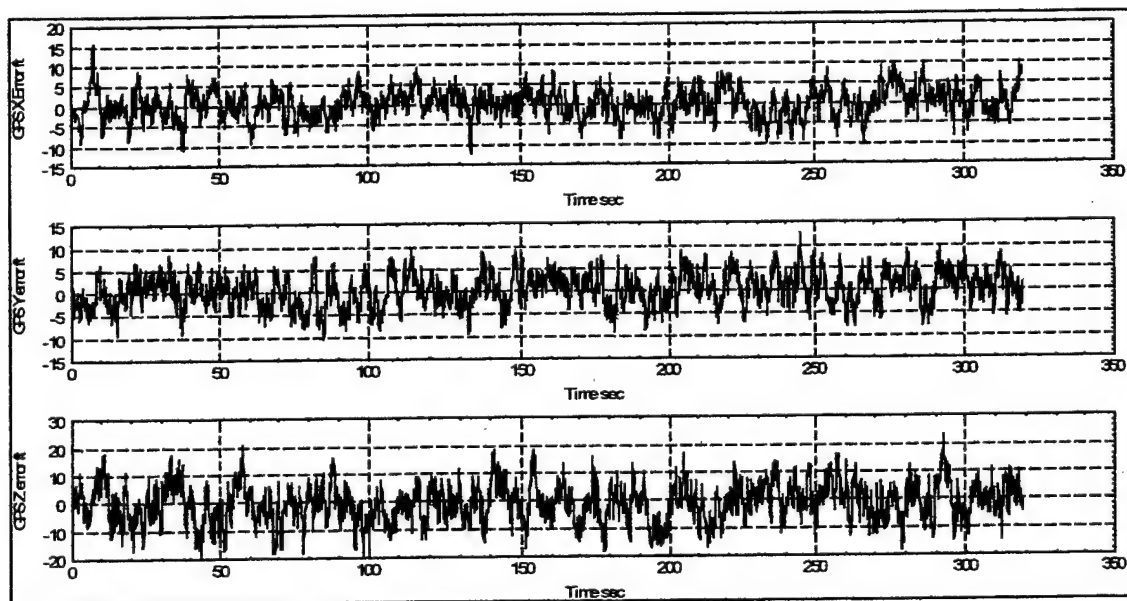


Figure 11. SystemBuild[®] GPS Errors

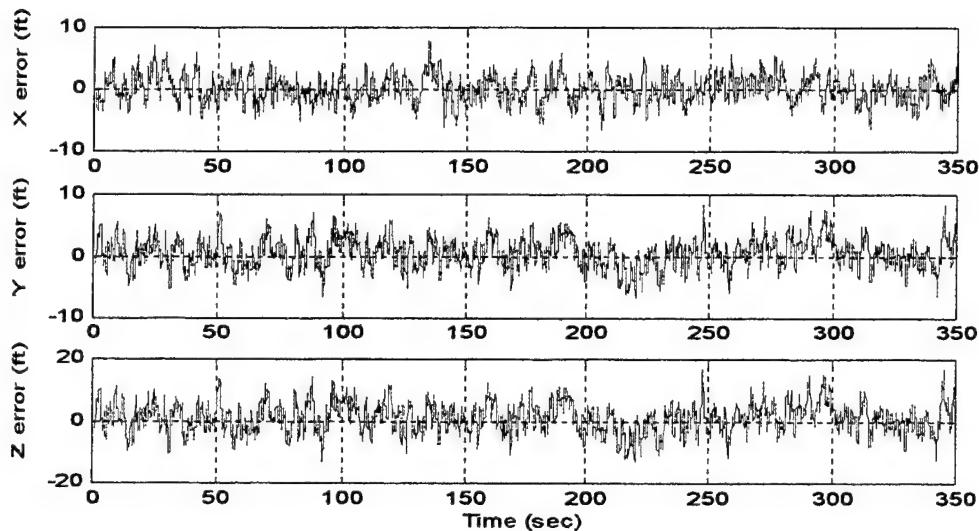


Figure 12. MATLAB® GPS Errors

The plot of GPS sensor position versus actual position for a typical simulation with selective availability off is not needed. Such great accuracy in the selective availability off receiver merits that the two are indistinguishable on such a plot.

2. Heading Sensor/Compass

The Attitude Heading Reference System (AHRS) is the magnetic compass modeled in this project. This system usually provides a static error component of ± 2 degrees and ± 1 degree with wind velocity aiding. A dynamic component of ± 2 percent is also present in this system, and similar compass systems show comparable results. Figures 13, 14 and 15 describe the XMATH® model of the AHRS used in this study, a graph of typical heading error as simulated by this model, and a graph of heading error with the MATLAB® model.

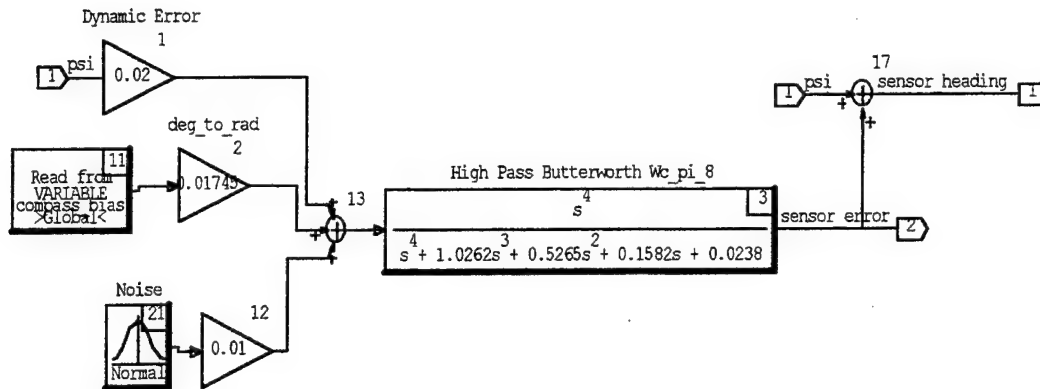


Figure 13. Heading Sensor Model

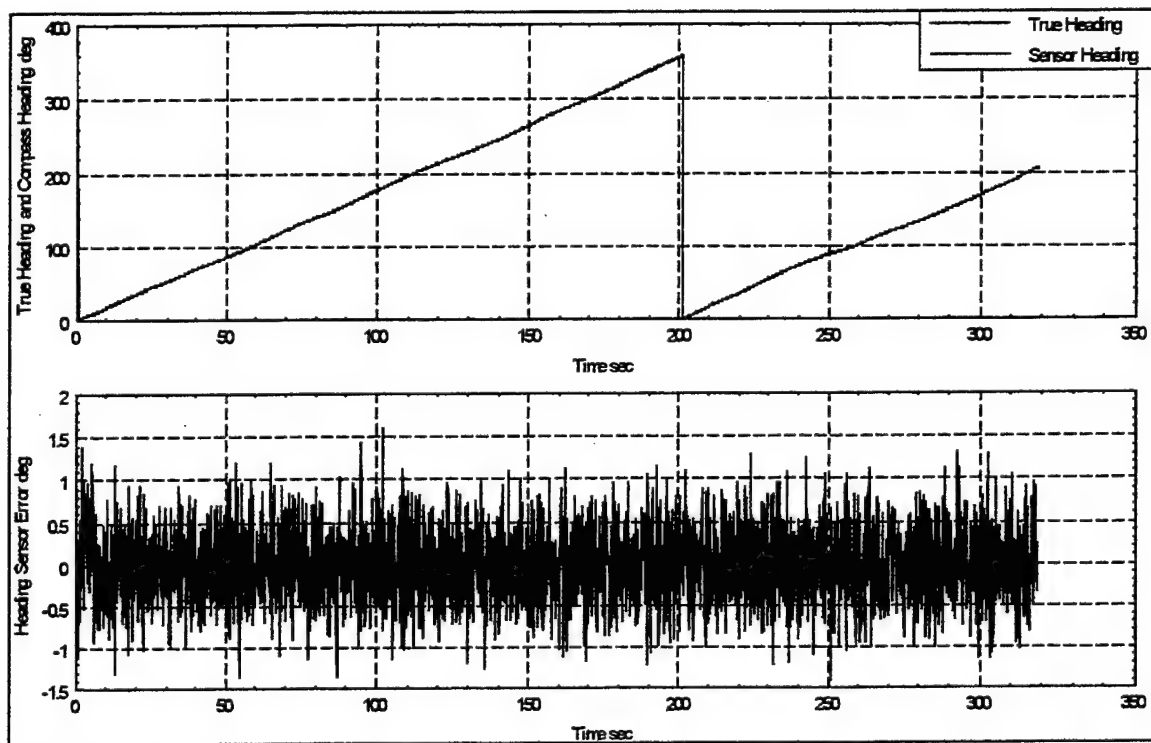


Figure 14. XMATH[®] Modeled Heading Sensor Error

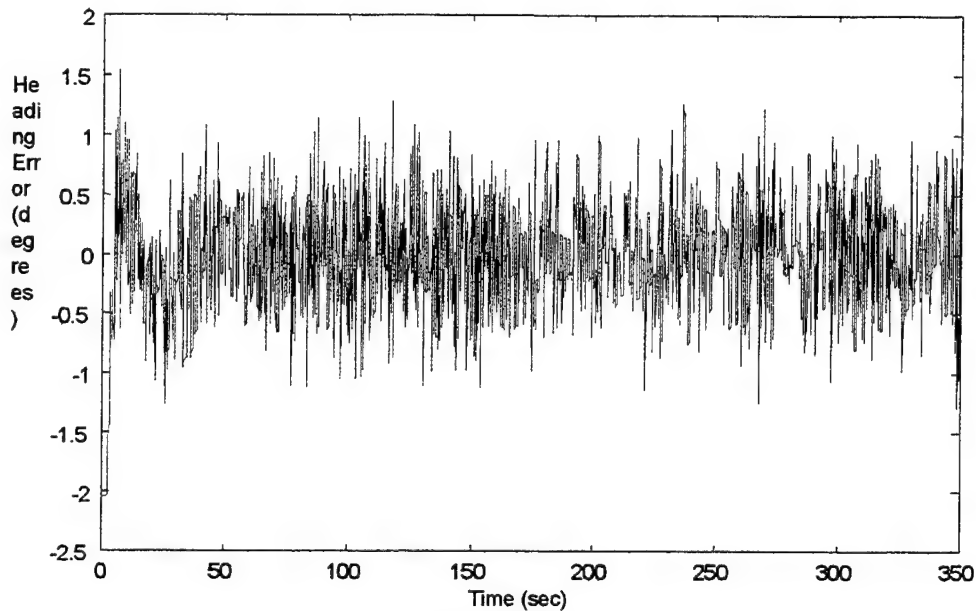


Figure 15. MATLAB[®] Modeled Heading Sensor Error

As a consequence of the GPS and heading sensor models, noisy x , y , and z positions and noisy heading (ψ) are outputs of these models. Since ϕ and θ are assumed zero throughout the parachute simulations, they are not measured and therefore there are no pitch or roll angle sensors. The transition of the model to six degrees of freedom would incorporate these sensors.

C. CONTROL SYSTEM MODELING

1. Control Strategies

The control system implements the control algorithm for the simulation. The objective of the parachute is to land at a target position, or to within 100 meters of this target position (approximately 300 feet). Probably the biggest unknown in the control of the parachute to the target position is the wind. The control system seeks to overcome

the wind and land the parachute in the correct position; however, the velocity of typical winds is much higher than the actuation system (the PMAs) of the parachute can overcome. Flight tests have shown that the most the PMA system can overcome is approximately a twelve-foot-per-second wind. Therefore, the control system can either steer the parachute toward a trajectory based on predicted winds, or ignore the prediction of the wind and head toward the final target position at all times.

The predicted trajectory used by the control system is determined by a previous simulation based on the most recent wind data, most likely collected by a Radiosonde Wind Measuring System (RAWIN). For this model, the simulation based on this wind data is known as a Computed Air Release Point (CARP) simulation. The CARP simulation is the same as the vehicle model described previously. It is just a description of the parachute's dynamics subjected to the predicted winds from the RAWIN balloon. There are no controls (set by a super block called "Null Controller") and the CARP predictor does not take into account the initial velocity of the parachute based on the aircraft's velocity. The CARP predictor is always released from the point [0; 0; -9500] (x, y, and z position in NED coordinates). Otherwise, the vehicle models are the same (same constant 1.8 degree per second rotation rate, 0 pitch and roll, same equations of motion). The final position of this CARP parachute when it hits the ground becomes the target position of the actual simulation. In final implementation, a target position on the ground for the actual parachute would be determined in advance, and the CARP simulation would then figure a predicted trajectory of the parachute based on the predicted winds from this target position, in effect determining the ideal release point for

the parachute. However, for simplicity the CARP is always dropped from the same position, and its landing point becomes the target. Random offsets from this ever-constant initial CARP position are chosen for the actual parachute simulation.

For the trajectory-seeking controller, the parachute follows the CARP's path at every altitude station throughout its entire drop. For the target-seeking control strategy, the parachute aims at the final position of the CARP (the target position) at every altitude station throughout its entire drop. This is the only difference between these two control strategies.

2. Parachute Control Logic

Figure 16 shows the flow diagram for the parachute controller. The controller determines which of four PMAs arranged in four axes (90 degrees apart) to activate. An activation of a PMA causes a movement of the parachute in a certain direction. This direction is toward the predicted trajectory determined by the CARP simulation or toward the target position on the ground at all times, depending on the control strategy used. The logic for the controller is as follows: sensed position from a GPS receiver is fed into a linear interpolation block that extrapolates the target position in x and y universal coordinates. The target position is the CARP trajectory determined from predicted winds and loaded into the model before the actual simulation, or the target position (also determined from CARP) at every altitude station (the controller aims at this target position at all times). This target is also known as the reference input. It is called "predicted_x" and "predicted_y" in the trajectory-seeking control strategy simulation and "target_x" and "target_y" in the target-seeking control strategy simulation.

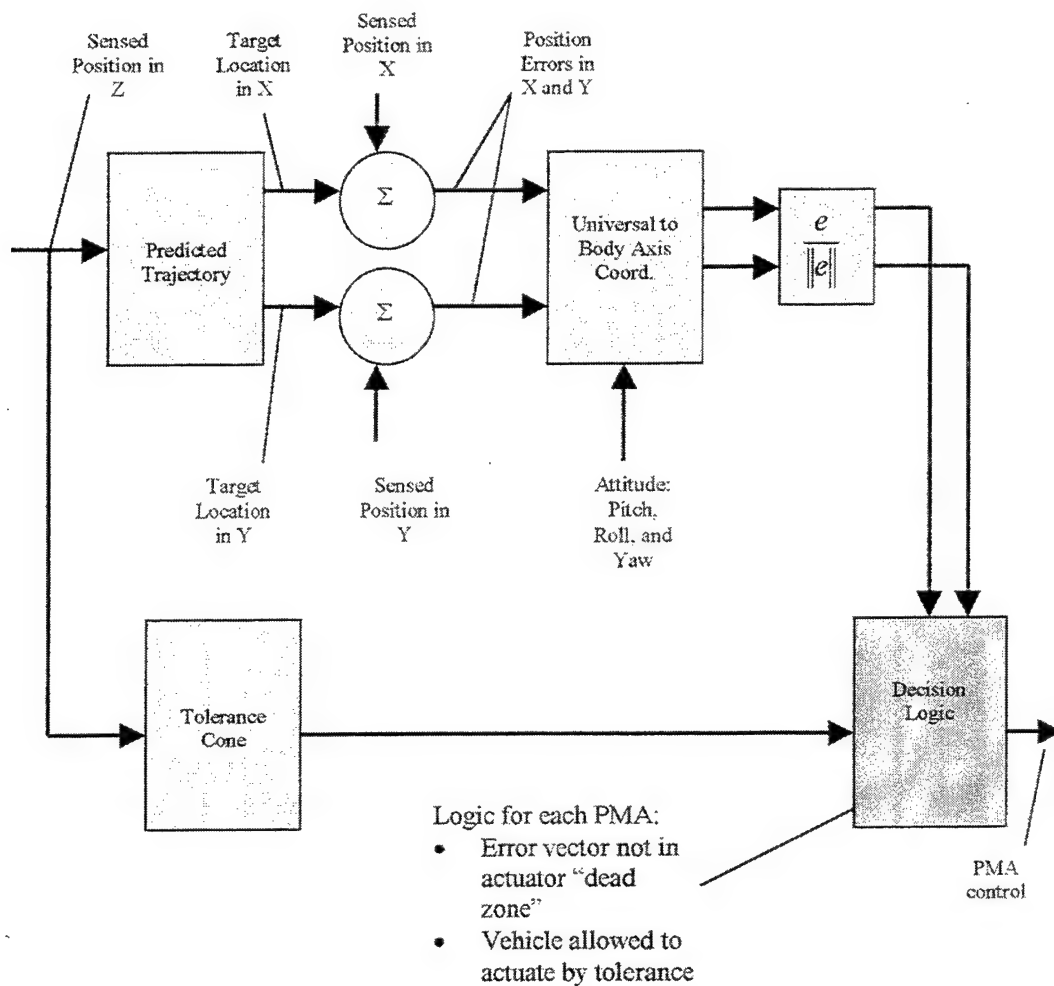


Figure 16. Flow Diagram for Parachute Control Logic

An error in x and y is determined from this reference input by subtracting the feedback signal of sensed position in x and sensed position in y ($e = r - y$ in standard control theory) [Ref. 11]. The errors in x and y are then converted to body-axis coordinates in order to take into account the rotation of the parachute and the position of the PMAs relative to the line of sight of the predominant error. The body-axis errors in x and y are each divided by the radial error (just the norm of these body-axis errors in x and

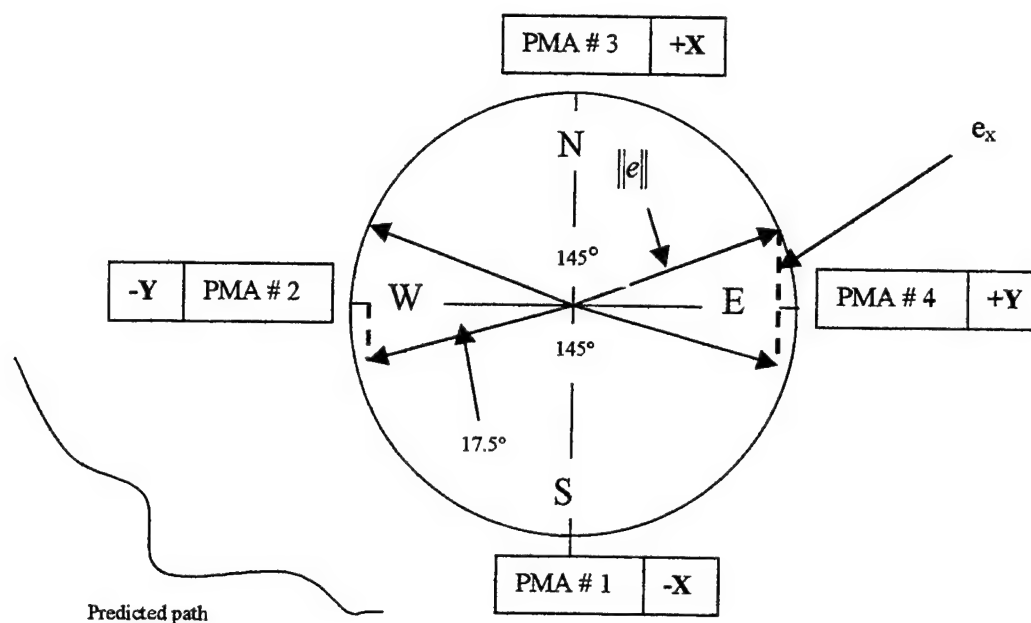


Figure 17. Operating Angles

y) in order to transform the body-axis errors into an error in the “operating angle” of each of the PMAs. A diagram is incorporated to better explain this concept.

Figure 17 shows the arrangement of the PMAs on the body-axis of the parachute. Remember, north is in the positive x direction, and east is in the positive y direction in the north-east-down (NED) coordinate system.

The 145° angle opposite PMAs 3 and 1 in Figure 17 are the operating angles of those PMAs. All four PMAs have operating angles of 145°. The reason for this is as follows: suppose the trajectory the parachute is supposed to follow (the predicted trajectory from CARP) has a line of sight of approximately 225° (in the southwesterly

direction) as shown in the diagram. This is the reference trajectory and the calculation of the error in x is negative. Next, the radial error to this reference trajectory is calculated and the error in x is divided by this radial error. If this calculation of $\frac{e_x}{\|e\|}$ is negative and less than -0.3, then PMA #3 is activated. An activation of PMA #3 will drive the parachute in the negative x direction, toward the predicted path. This is because an activation is a VENT of a PMA. A vent of a PMA causes a lengthening of a parachute riser, which in turn causes a spillage of air on the side of the vented PMA. This creates a drive in the opposite direction.

If PMA # 3 is activated on a $\frac{e_x}{\|e\|} < -0.3$, this corresponds to a "dead zone" where the PMA is left idle of approximately 17.5° on both sides of the negative x half of the circle. This leaves an angle where the PMA operates at approximately 145°. Similar operating angles are calculated for the other PMAs.

The logic performed to determine if a PMA needs to be possibly activated (vented) depends on whether the predominant error is in the PMA's operating angle:

- if $\frac{e_x}{\|e\|} < -0.3$, error is in PMA # 3 operating angle
- if $\frac{e_x}{\|e\|} > 0.3$, error is in PMA # 1 operating angle
- if $\frac{e_y}{\|e\|} < -0.3$, error is in PMA # 4 operating angle

- if $\frac{e_y}{\|e\|} > 0.3$, error is in PMA # 2 operating angle

These logic checks are performed by logical expression blocks in SystemBuild®.

If the above expressions are true, a 1 is output from these logical expression blocks (corresponds to TRUE).

However, there needed to be constraints on the amount of radial error that merited a control activation. If there were no constraints, the PMAs would be constantly activating, which was not feasible. Therefore, a “tolerance cone” was built to apply these error constraints. Figure 18 contains an example of how this tolerance cone worked,

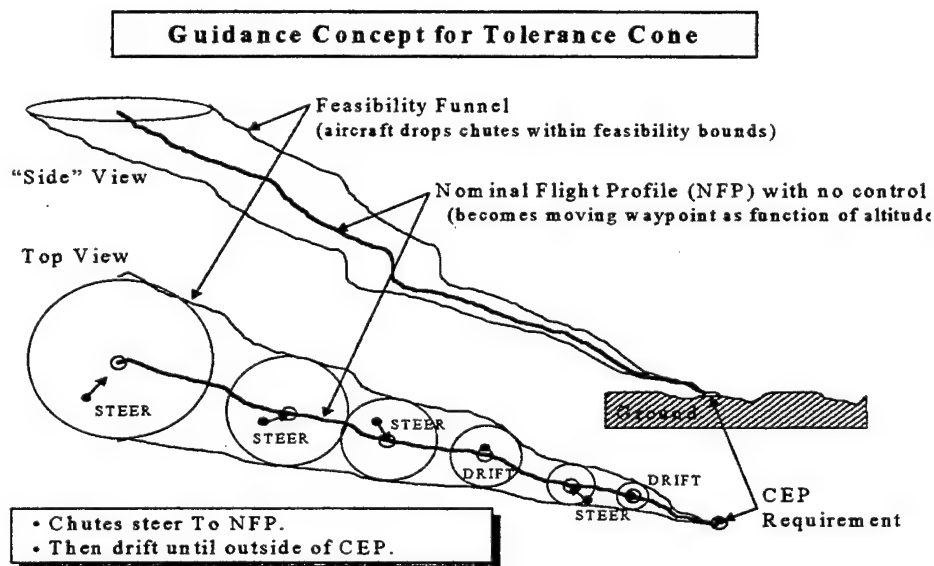


Figure 18. Tolerance Cone Guidance Concept From Ref. [12]

taken from a presentation on the subject by the Boeing Corporation. In the diagram, the NFP stands for Nominal Flight Profile. This is a circular "tube" surrounding the predicted path (a.k.a. reference path) at every point in the path's trajectory. This tube never changes with altitude. The CEP (Circular Error Probable) DOES change with altitude. It is an outer cone for the guidance concept. The largest part of the cone is at the top where there is a greater tolerance for error. This brings up the concept of the "feasibility funnel". The "feasibility funnel" is a cone where, at each altitude in the cone, the circle in the horizontal plane describes an area out of which the aircraft cannot drop the parachute because it would never make it into the desired final CEP from outside this area. This area is totally determined by the drive of the parachute actuators. It assumes an exact wind prediction and zero mean random GPS and heading errors. In the Boeing diagram, the "feasibility funnel" describes the tolerance cone. However, this does not have to be the case. In the case of this study, the tolerance cone's outer shell (the CEP) is an "upside-down wedding cake". It is formed by a block script in XMATH®, with altitude as the input (a positive altitude in NEU coordinates) and the following code:

```
inputs: u;
outputs: y;
float u, y;
if u>8000 then
    y=500;
elseif u>6000 then
    y=400;
elseif u>4000 then
    y=300;
elseif u>2000 then
    y=150;
else
    y=60;
endif;
```

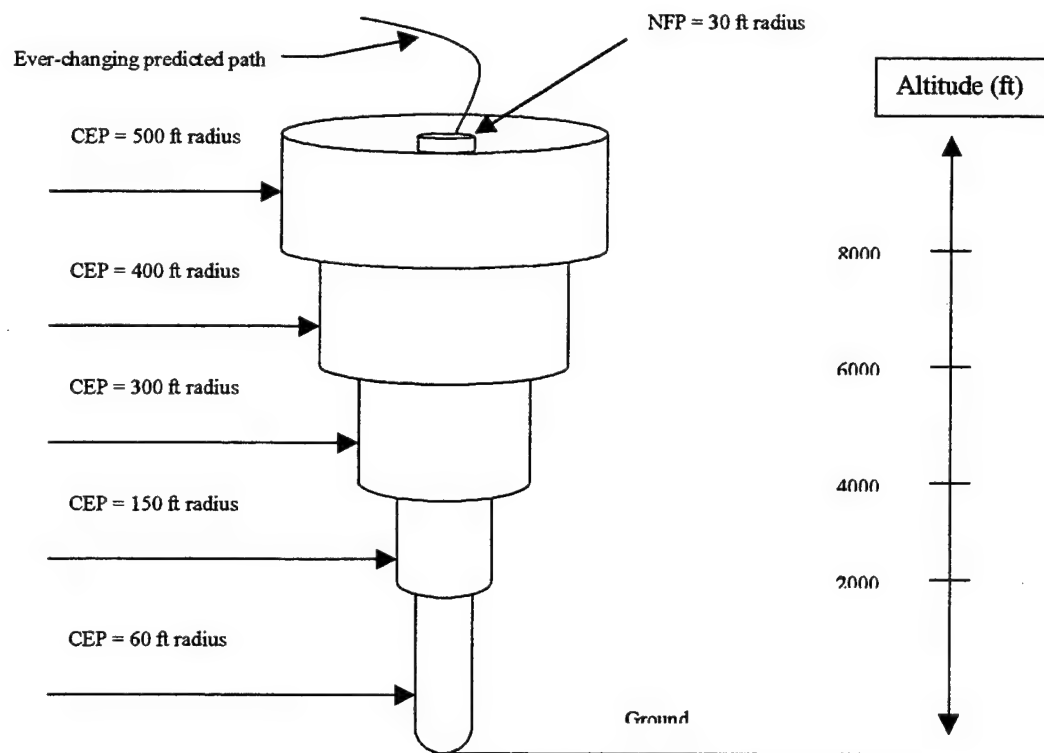


Figure 19. Tolerance Cone used in Simulation

All values are in feet. The inner shell of the tolerance cone (NFP) is always set at 30 feet. This creates a tolerance cone that looks somewhat like the diagram in Figure 19 and follows the predicted path at every waypoint. With the tolerance cone's inner and outer shells set, the logic of the guidance concept is as follows:

- if the radial error is outside the NFP, the control system steers the parachute until it is inside the NFP
- the parachute is then allowed to drift until it is outside the changing-with-altitude CEP

- once the parachute is outside the CEP, it is again steered until it is inside the NFP.

The decision-making process is clearly dependent on the state of the parachute's error. Therefore, a tolerance cone state diagram needed to be built in SystemBuild[®]. The state diagram for the tolerance cone is shown in Figure 20.

The inputs to the state diagram are the radial error (U1) and the outer shell of the tolerance cone (U2). The dark arrow at the top of the diagram indicates the bubble the state diagram checks first. The state first enters the "controlled" bubble. If the radial error is greater than 30 ft. (the NFP), the state continues to be in the "controlled" bubble, as indicated by the arrow flowing from and to the "controlled" bubble.

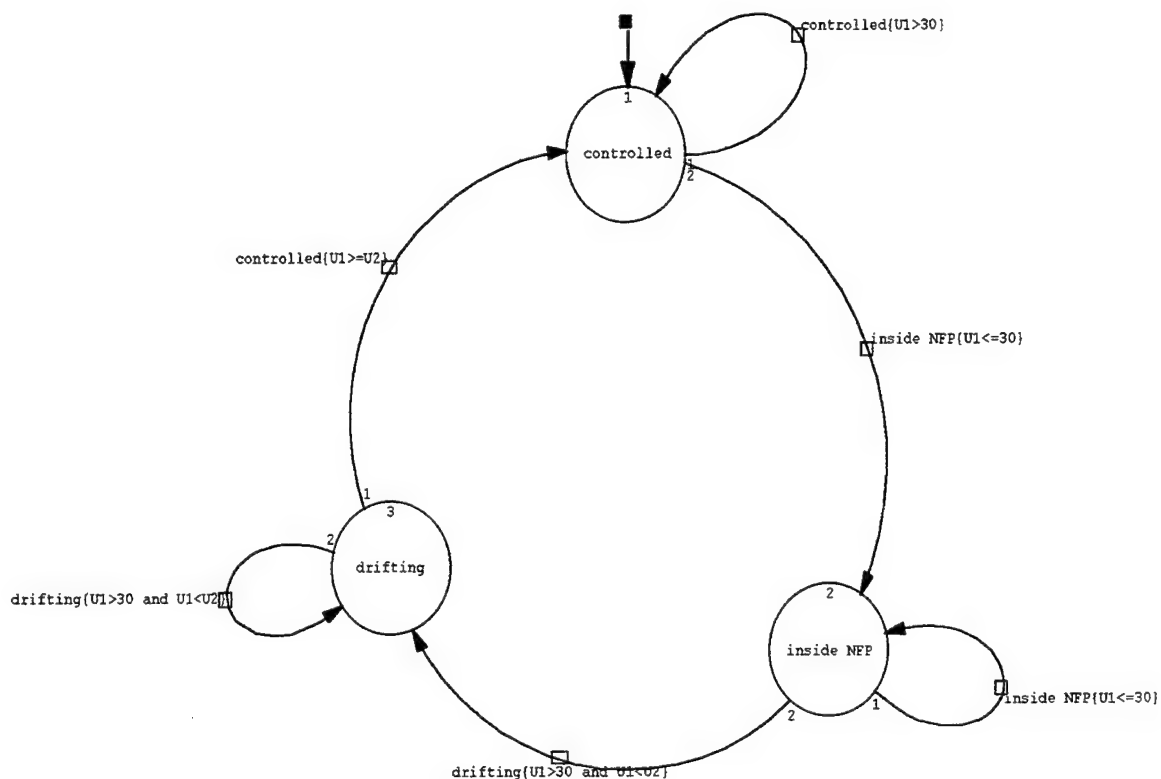


Figure 20. Tolerance Cone State Diagram

As the parachute is controlled and the error becomes smaller and smaller, the parachute goes inside the NFP, as indicated by the arrow flowing from the "controlled" bubble to the "inside NFP" bubble. The condition for this transition is if the radial error is less than or equal to 30 feet. As long as this is the case, the state stays in the "inside NFP" bubble and a 0 is output. If the radial error goes outside the NFP but is still inside the CEP, the state transitions to the "drifting" bubble, indicating that the parachute is drifting in between these two "tubes". The condition for this transition is if the radial error becomes greater than 30 feet but still less than the CEP (input U2). As long as this is true the state stays in the "drifting" bubble and a 0 is output. Next, if the radial error becomes greater than or equal to the CEP, the state once again transitions to the "controlled" bubble, and the process repeats. So, a 1 output from the state diagram means the parachute needs to be controlled toward the predicted path, and a 0 indicates that the parachute need not be controlled.

In order for the control system to activate a control, the body axis error must be in the PMA's operating angle, and the tolerance cone state diagram must allow a control input (see Fig. 16). Both the logical expression block that determines if the error is in the operating angle and the state diagram output a 1 or TRUE value if these questions answer true. Therefore, a simple 2-input AND logical expression with these two outputs as inputs to the AND block will determine if a control is on or off (1 for on, 0 for off).

Figure 21 shows the realization of the controller in SystemBuild®. Once it is determined whether or not a PMA will be actuated or not, these signals must be converted to actual commands in psi. Remember, an actuation is a venting of the PMA,

so this is a command to send the PMA pressure to 0 psi. A 0 corresponds to taking the actuation off, or filling the PMA, so this is a command to send the PMA pressure to whatever the maximum pressure of the actuators is. For this study, the maximum PMA pressure ranged from 100 psi to 175 psi. In the SystemBuild® diagram, this number is set through a parameter called “pma_max_pressure”. The PMA command pressure is set through a simple algebraic expression:

$$(7) \quad PMA_cmd_psi = pma_max_press. - pma_max_press. \times (pma_on_off)$$

where pma_on_off is equal to the 1 or 0 value corresponding to the PMA being actuated or turned off. Thus, the outputs of the controller are these PMA commands in psi, which become inputs to the PMA system model. The number of control actuations is also counted using an algebraic expression. This sets the drive of the actuators, which is dependent on the number of actuations at one time (based on flight tests).

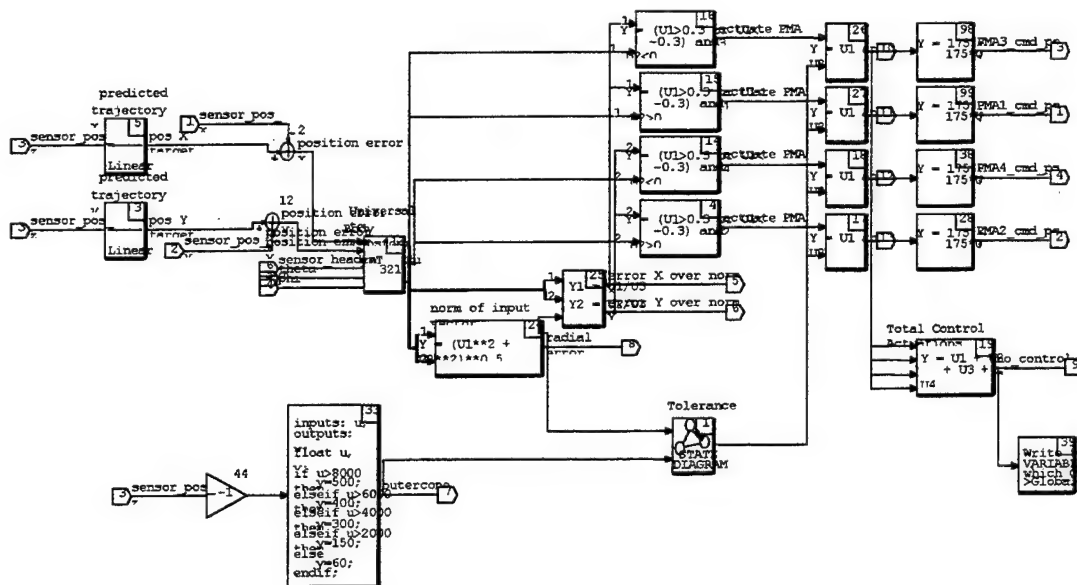


Figure 21. Controller

Figure 22 is a good example of the controller's performance. In Figure 22, the parachute has a perfect wind prediction loaded into CARP (both the CARP and the actual simulation have the same wind profile). The top two plots are plots of $\frac{e_x}{\|e\|}$ and

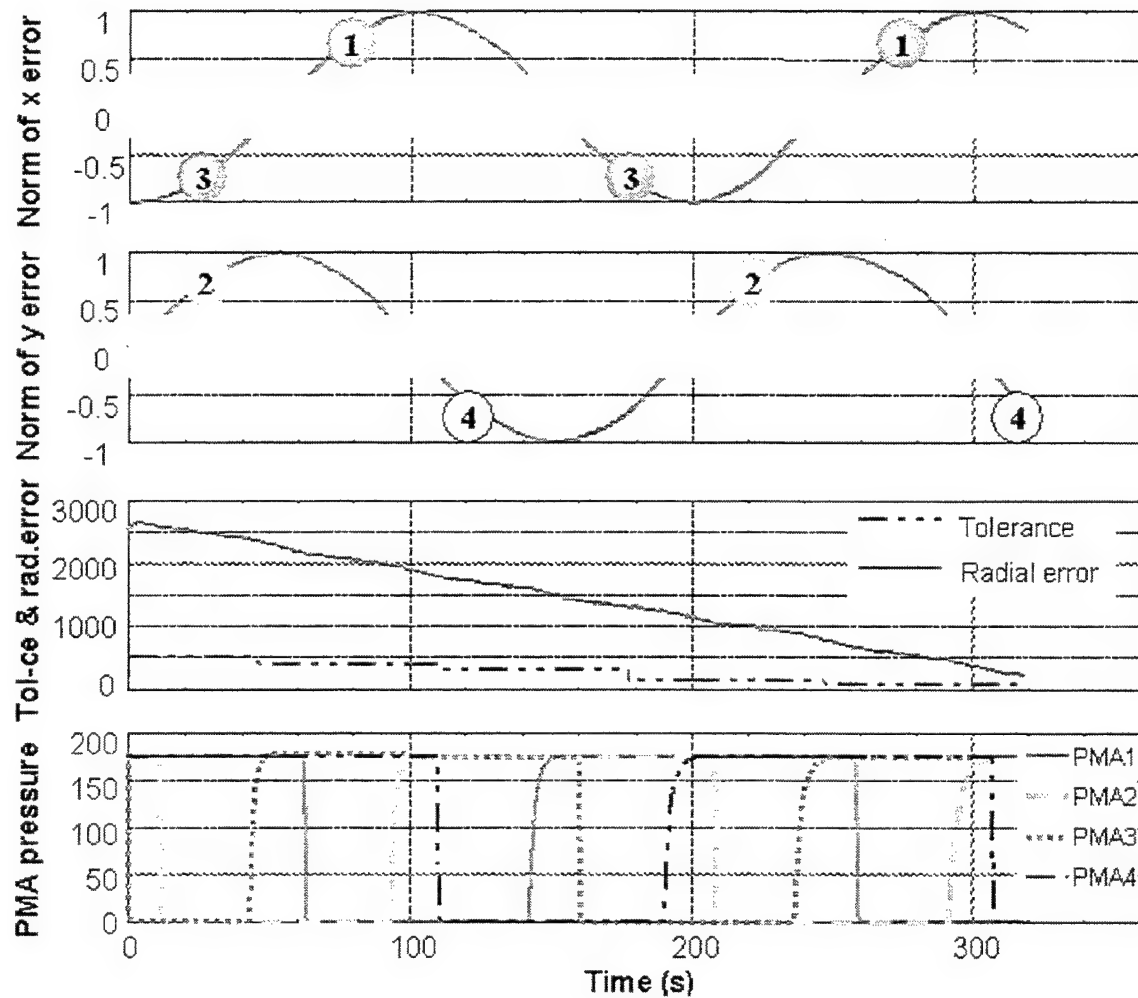


Figure 22. Control Logic with Good Wind Prediction

$\frac{e_y}{\|e\|}$. Initially, the parachute drop point is offset in the x direction by 2500 feet (so the initial x error is -2500 feet). Because of this fact, the constant rotation rate, and the perfect wind prediction, the actuators drive in one direction (toward the negative x or south direction) as the parachute rotates, and the plots of $\frac{e_x}{\|e\|}$ and $\frac{e_y}{\|e\|}$ are smooth sine functions. This constant driving direction is shown in the third plot of the tolerance cone and the radial error. This plot also shows the so-called “feasibility funnel”. The parachute is offset by 2500 feet from the ideal drop point and just barely makes it to the final CEP with an exact wind prediction. This makes 2500 feet the approximate radius of the “area of attraction”. This area was determined using the script file “attraction”. The fourth plot shows the actuation history of the four PMAs. One can observe that the PMA activations follow the control logic described above. The circles with numbers mark the points on the first two plots where a certain PMA was activated.

D. ACTUATOR SYSTEM MODELING

Dellicker’s thesis modeled the actuators as instantaneous control inputs, meaning the model did not reflect the dynamic characteristics of the actuators, including valve opening and closing times, filling and venting dynamics, and control force coefficients of the actuators. This study sought to model these characteristics and assess the results of this actuator behavior. Much testing has been done in an attempt to characterize PMA behavior. The basic premise of the actuator model is to re-create this test data in real time as the simulation is running in order to best describe actuator dynamics.

Figure 23 shows a diagram of the actuator setup in the parachute payload from a presentation by Vertigo, Incorporated, the makers of the PMAs. The gas for filling the actuators comes from 4500 psi reservoirs (the diagram shows two, but in the simulation for this study, only a single 4500 psi reservoir is used). Each of the four actuators are then connected to this same reservoir of nitrogen gas through some piping or tubing leading to a fill valve. The fill valve is opened to allow gas to fill the actuators when a command to take an actuation off is received. When the pressure inside the PMA reaches a certain value, a pressure switch signals the fill valve to close.

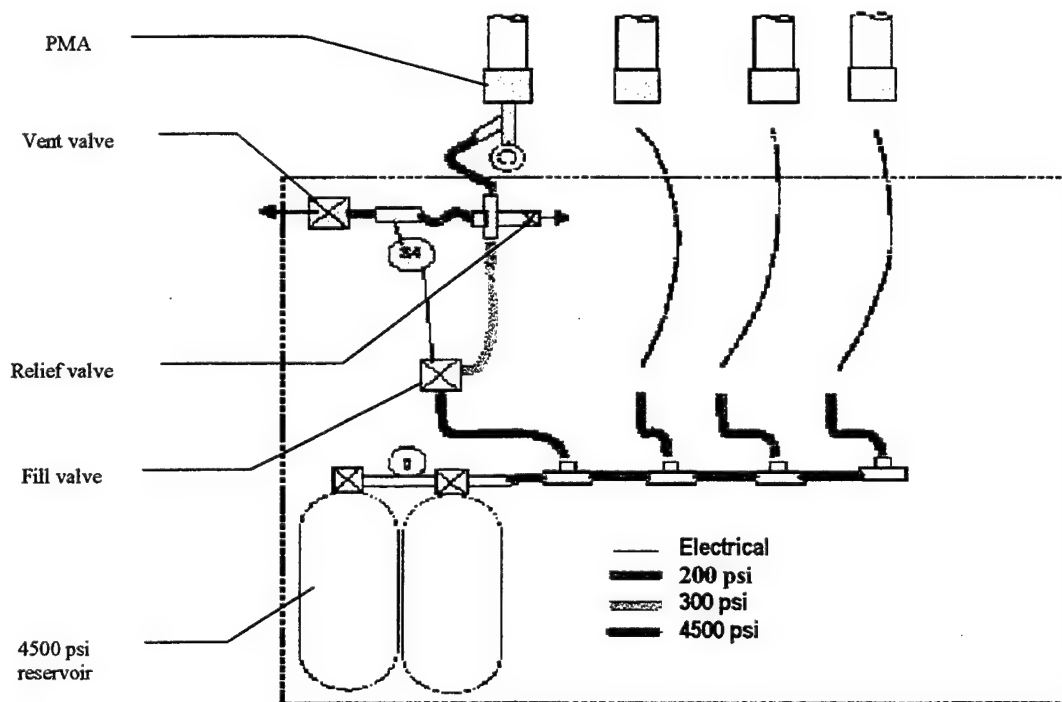


Figure 23. Vertigo, Inc. Actuator System Concept From Ref. [13]

Since the fill valve works with high-pressure gas it has a small orifice and therefore opens and closes rather quickly upon receiving the correct electrical signal. The time to open and close the valve is roughly 100 ms. However, the decrease in pressure of the gas tank as more and more fills are completed slows down the actual filling process. This dynamic characteristic is modeled in XMATH[®] based on collected actuator data.

The vent valve opens to empty the actuator when a command to actuate is received. The vent valve has a large orifice and can open quickly to vent the PMA, but requires a certain time to vent the gas and close the orifice. Each opening of the vent valve requires approximately 100 ms, but the venting process and closing of the valve depends on the maximum pressure of the actuator fill. This process also takes a constant amount of time (approximately) because the pressure in the actuators is the same upon

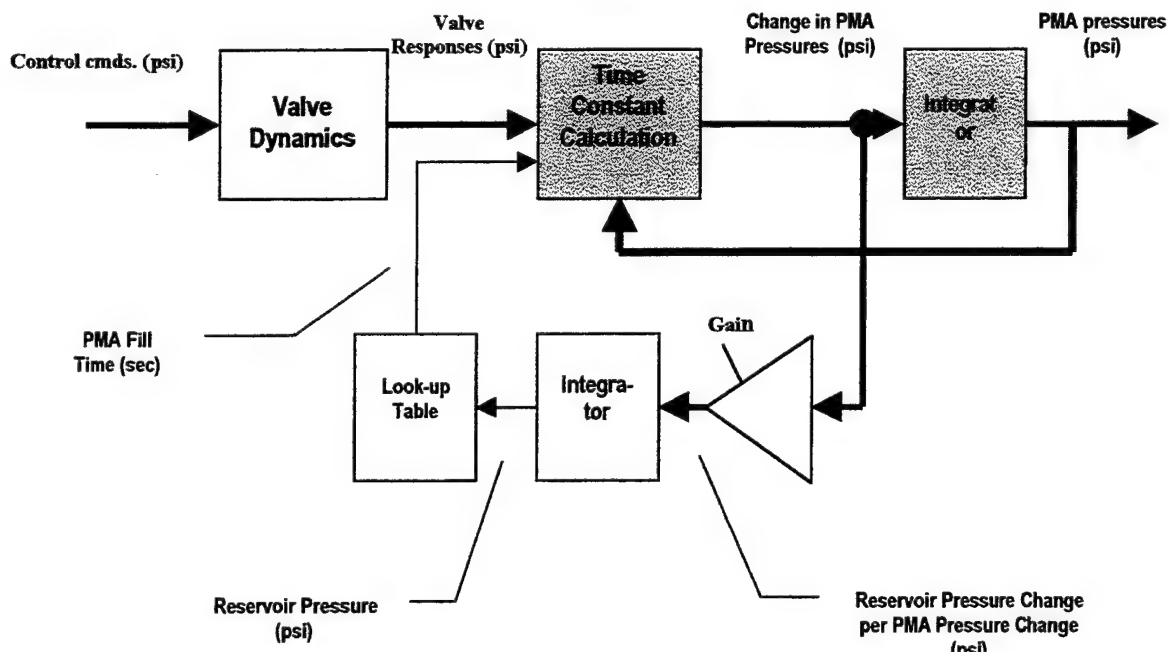


Figure 24. Actuator Modeling Concept

each vent. Many tests have been run on both the filling and venting process of the actuators, but for now only the modeling is described here.

A basic diagram of the actuator modeling is shown in Figure 24. The four commands from the controller in psi (either 0 or the maximum actuator pressure) are fed into a block called "Valve Dynamics". This block models the opening and closing of the valves for both filling and venting. This process takes approximately 100 ms. In these simulations, the opening and closing of the valves was taken to be a worst-case 166 ms (approximately) and was modeled as a first order lag with a time constant of 0.0333 seconds. It takes about five time constants to reach the final value of a first order lag, so $5 \times 0.0333 = 0.1665$ seconds is a rough estimate for the opening and closing of the valve.

The Laplace equation for a first order lag for one PMA is:

$$(8) \quad X(s) = \frac{1}{\tau s + 1} U(s)$$

where $X(s)$ is the Laplace of the output $x(t)$ (the result of the valve dynamics), $U(s)$ is the Laplace of the input $u(t)$ (the commands), and τ is the time constant. Expanding this Laplacian equation out to get it in differential equation form (for state-space methods):

$$(9) \quad (\tau s + 1)X(s) = U(s)$$

$$(10) \quad sX(s) + \frac{1}{\tau}X(s) = \frac{1}{\tau}U(s)$$

$$(11) \quad \dot{x}(t) - x(0) + \frac{1}{\tau}x(t) = \frac{1}{\tau}u(t)$$

$$(12) \quad \dot{x}(t) = -\frac{1}{\tau}x(t) + \frac{1}{\tau}u(t) + x(0)$$

This final differential equation is then put in state-space form for an XMATH[®] state-space block. The block needs specified initial conditions, $x(0)$, and the matrices A, B, C, and D as follows:

$$(13) \quad \begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

where \dot{x} , x , y , and u are all 4x1 vectors (1 element for each PMA and y is the output vector) and A, B, C, and D are 4x4 matrices describing the above differential equation [Ref. 11]. With a time constant $\tau = 0.0333$ seconds, the matrices become:

$$(14) \quad A = \begin{bmatrix} -30 & 0 & 0 & 0 \\ 0 & -30 & 0 & 0 \\ 0 & 0 & -30 & 0 \\ 0 & 0 & 0 & -30 \end{bmatrix}; B = \begin{bmatrix} 30 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 \\ 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 30 \end{bmatrix}; C = I(4 \times 4); D = 0;$$

With an initial condition for each of the PMAs being filled (at the maximum actuator pressure), the state space block is complete and the valve dynamics are roughly described.

The valve responses in psi become inputs to a "Time Constant Calculation" block.

This block models the PMA filling and venting by first order lags. This is accomplished through a block script with the following code:

```
inputs: (x, signal, time_fill, deflate_time);
outputs: (xdot);
environment: (INIT);
parameters: pma_max_pressure;
float signal(4), x(4), xdot(4), k(4), e(4), time_fill, tau,
pma_max_pressure, deflate_time;

tau = time_fill/5;
```

```

for i=1:4 do
e(i) = signal(i) - x(i);

if (e(i) >= 0.0) then
k(i) = 1/tau;
else
k(i) = 1/(deflate_time/5);
endif;

if (x(i)<=0 & e(i)<0) | (x(i)>= pma_max_pressure & e(i)>0)
then
xdot(i) = 0;
else
xdot(i) = k(i) * ( signal(i) - x(i) );
endif;

endfor;

```

The inputs to this code are x (the current state of each of the PMAs), $signal$ (the command signal after valve dynamics modeling), $time_fill$ (the fill time based on how much pressure is left in the reservoir) and $deflate_time$ (the constant vent time based on what the maximum actuator pressure is, set before the simulation starts). The output of the code is $xdot$, the change in PMA pressure per unit time for each of the PMAs. So, there are a total of 10 inputs (4 each for x and $signal$) and 4 outputs (4 for each PMA).

The code next calculates the value of τ , the time constant when a PMA is filling, by dividing fill time by 5, since the time to reach the final value of a first order lag (the fill time) is approximately five time constants ($5 \times \tau = risetime$). Then for each PMA, the code calculates a value for e . This value is the signal command (what the PMA should be) minus the current state of the PMA. If this value is greater than or equal to 0, the PMA is filling, and the variable k is set to $1/\tau$. If it is less than 0, k is set to $1/(deflate_time/5)$ where $deflate_time/5$ is the time constant if the PMA were deflating.

Next, an IF-THEN-ELSE statement sets the differential equation for the calculation of \dot{x} . This differential equation for a first order lag is the same as Eq. 12, but rearranged:

$$(15) \quad \dot{x}(t) = -\frac{1}{\tau}x(t) + \frac{1}{\tau}u(t) + x(0); \quad x(0) = 0; \quad k = \frac{1}{\tau}; \quad signal = u(t)$$

$$(16) \quad \dot{x}(t) = k(signal - x)$$

The IF-THEN-ELSE statement asks two questions:

- Is the state of the PMA less than or equal to 0 (it should not be less than 0) and is e less than 0, meaning the PMA is all vented and it is still being commanded to vent?
- Is the state of the PMA greater than or equal to the max pressure in the PMA (should not be greater than this max pressure) and is e greater than 0, meaning the PMA is all filled and is still being commanded to fill?

Based on the answers to these questions, the \dot{x} equation is formed. If either one of the two above questions is true, the change in pressure in the PMA, \dot{x} , is ZERO. Otherwise, the above differential equation applies with the correct value of k already calculated (depending on whether the PMA is filling or venting), and the code is complete.

The change in each of the PMA's pressures per unit time is then integrated (with initial condition being the PMA maximum pressure--recall that the PMAs start being all filled). The PMA pressures are then limited between 0 and the PMA maximum pressure by a SystemBuild® limiter block, and this becomes the state of each of the four PMAs. These states are the primary outputs of the PMA model.

The fill time of the actuators is calculated by data interpolation on the remaining reservoir pressure. Therefore, the pressure left in the reservoir needs to be calculated. This is done by taking the changes in pressure per unit time for each of the PMAs (\dot{x}), and feeding this through a gain reflecting the change in the reservoir pressure per a full PMA fill. This gain is actually a linear interpolation block with index being the remaining reservoir pressure. From remaining reservoir pressure one can interpolate how much gas will be expended from the reservoir on the next full PMA fill. This number is divided by the max PMA pressure to give the gain for this particular block. This change in reservoir pressure for that fill is divided by the max PMA pressure because the \dot{x} input into this gain is a PART of this maximum pressure that is summing up to the maximum pressure over time. The small parts of this PMA max pressure multiplied by the gain will slowly add up to the pressure expended from the reservoir for that PMA fill. After being multiplied by the gain, each of the expenditures from the reservoir (for each PMA) are then multiplied by -1 and summed up to obtain the total pressure used by the reservoir to fill up the PMAs.

The change in pressure of the reservoir is then integrated (with initial condition being the initial pressure of the reservoir) over time and remaining reservoir pressure in psi is output. Reservoir pressure becomes an input into the gain interpolation block discussed above and the fill time interpolation block. Fill time then becomes an input to the time constant calculation block. Completed fill cycles can also be interpolated from remaining reservoir pressure using a linear interpolation block. The complete PMA model used in simulation is shown in Figure 25.

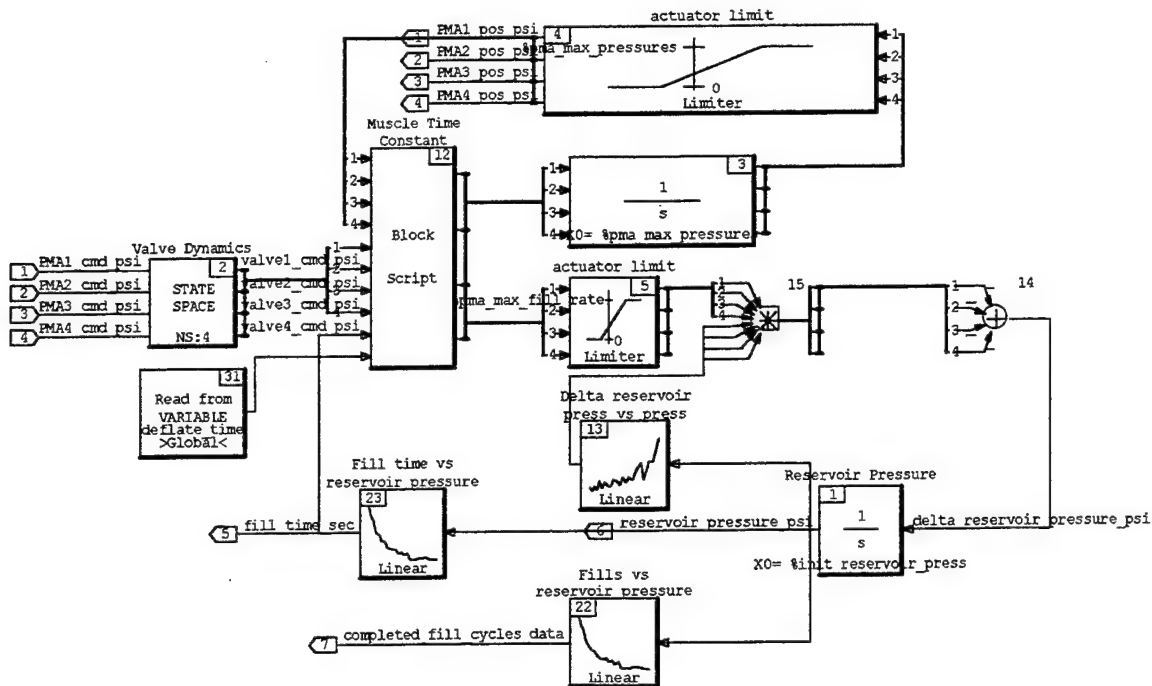


Figure 25. PMA Model

The PMA positions in psi become the inputs to the vehicle model described in the parachute system modeling. In that model, the PMAs provided an applied force on the parachute in a certain direction. This was accomplished in the "Force Rotation w/ Length" block. This model is shown in Figure 26.

The position of the PMAs is immediately transformed to a corresponding change in the length of the parachute risers through the linear interpolation block "Change in Length vs. Actuator Psi", the values of which are obtained from collected data. There are two channels of PMAs going through two of these type blocks, and then going through another linear interpolation block labeled either "One or Less Control Inputs Cf" or "Two or More Control Inputs Cf". The Cf of the PMAs is assumed to be dependent on the

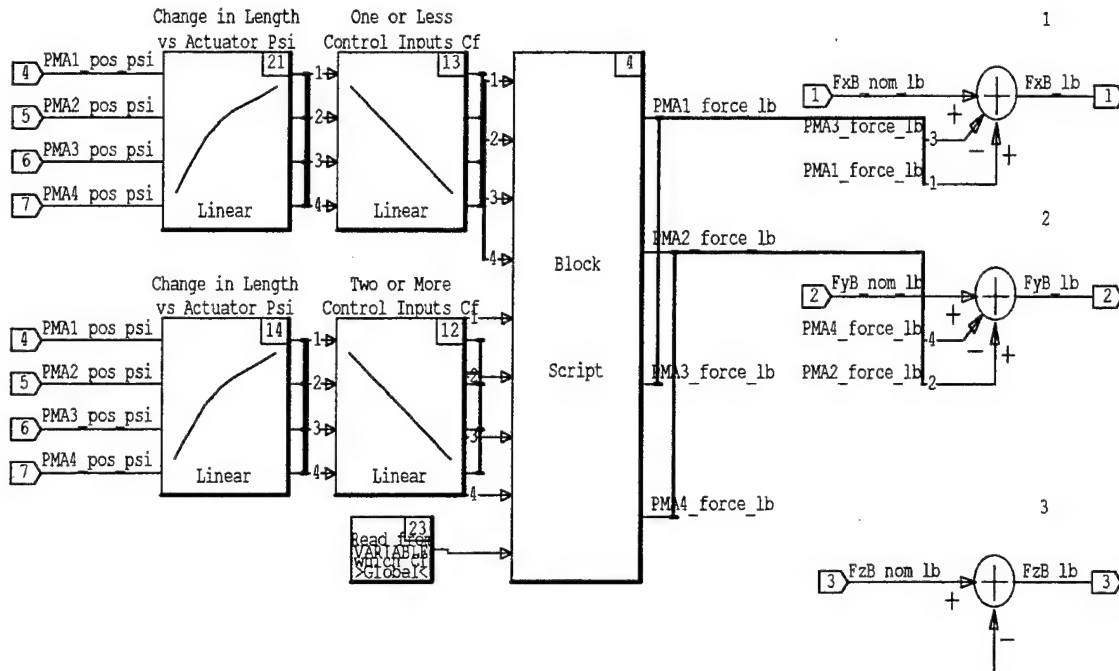


Figure 26. Force Rotation with Length Change

number of controls activated at one time. This assumption was made from the flight test data in Dellicker's thesis. He gathered from this data that when there was only one control input the glide ratio of the parachute was approximately 0.4, and when there were 2 control inputs the glide ratio was approximately 0.2. Further flight test data could prove this assumption wrong, but for this project the assumption was used. On a full riser length change caused by one PMA, the glide ratio was assumed to be 0.4 and the force coefficient (or added force in pounds) was determined by trial and error to be 900 lbs. The same process was used to find the force coefficient for two full riser length changes caused by two PMA control inputs and was found to be 317 lbs. The block script block merely allows the 0.4 glide ratio force coefficient when there are one or less control inputs and the 0.2 glide ratio force coefficient when there are 2 or more control inputs

(there should never be more than 2 control inputs at one time). A graph of the changing glide ratio with number of controls is shown in Figure 27. [Ref. 8]

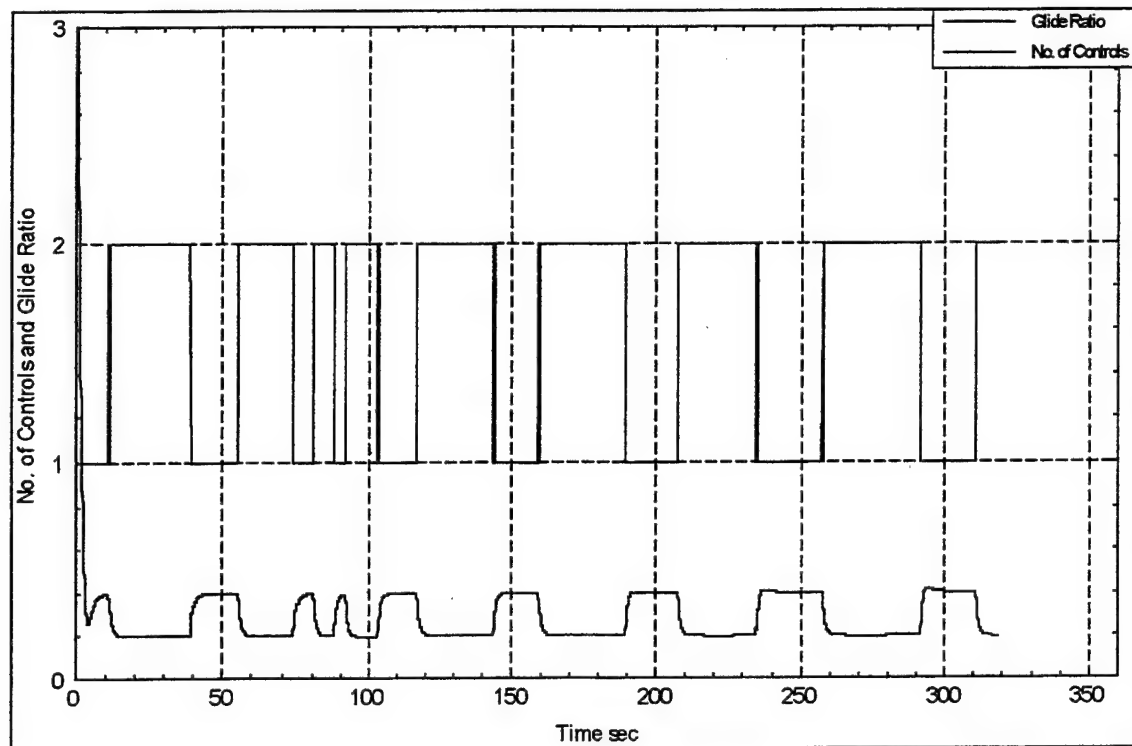


Figure 27. Number of Controls and Corresponding Glide Ratio

The forces applied by the PMAs are then added to the nominal forces (the drag on the parachute) in the x and y directions to affect a movement of the parachute in the correct direction. This is the direction opposite the side the PMA is on. No force is added by the PMAs in the z direction. Further study could provide some information as to a force in this direction. Additional study could also better characterize the actual affect of the PMAs beside an applied force, possibly a rotation of the drag vector or added moments.

It was found during testing and simulation that useful information might be gathered from counting the number of completed fill cycles during each run. This information might lead to possible design specifications on the actuators. However, counting the number of fill cycles was a complex process. One way was to use a linear interpolation table as described above. Another way was to use a state diagram to keep account of the state of the PMAs and only increment counting when a fill was commanded.

The state diagram "Counter" is shown in Figure 28. The input to this state diagram (U1) is the on/off command of a PMA to activate or deactivate in the "Controller" super block. A 1 is the input if the PMA is commanded to be on, and a 0 is the input if the PMA is commanded to be off.

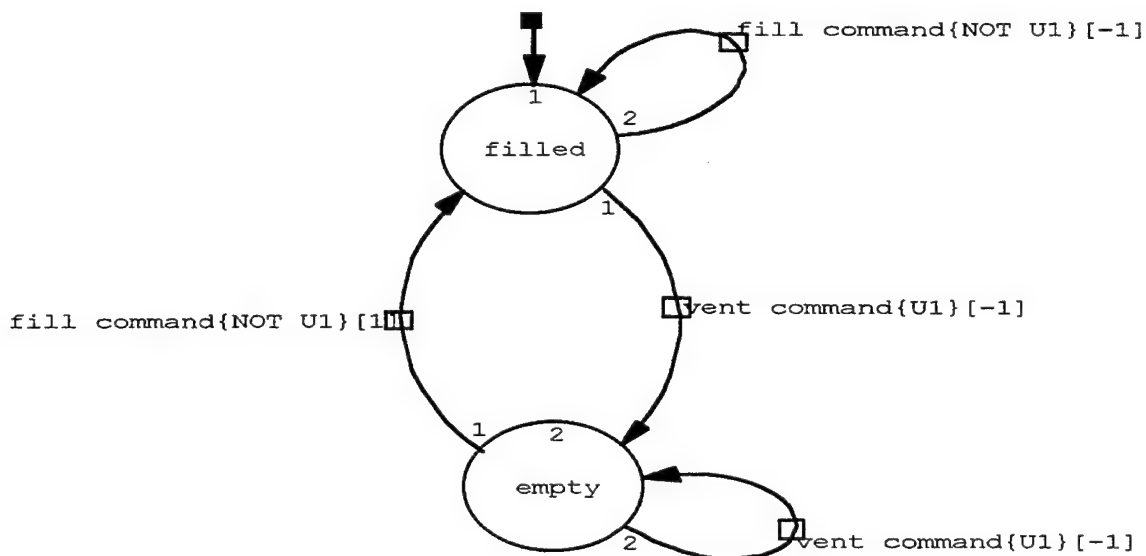


Figure 28. Counter

The input enters the state diagram in the filled state bubble. The conditional statement for the transition from the filled state bubble and back to it again (a loop) is NOT U1. Any time a conditional statement is TRUE, the transition is made and the Mealy output for that transition becomes the output of the state diagram. In this example, the command to fill is a 0; NOT U1 is 1, or TRUE, so the transition is made back to the filled state bubble and a 0 (denoted by -1) becomes the output of the state diagram. If a command to vent the PMA, a 1, is given while the state is in the filled state bubble, a transition is made to the empty state bubble and a 0 becomes the output. A loop is also made for the vent command while this command continues to be made. If a command is given from the empty state bubble to the filled state bubble (a fill command), then the transition is made to the filled state bubble and a 1 becomes the output of the state diagram. This is the only time at which a PMA fill is counted. The cycle then repeats. Counters for each of the PMAs were built and the outputs of these state diagrams were added together, with a recursive loop created by a data store block in SystemBuild®.

The state diagram encountered some problems, however. Because of fluctuations around the actuator "dead zones" and the tolerance cone caused by GPS and heading sensor error, the command to fill or vent a PMA often times would be unstable at these points. In this case the counting of COMMANDS to fill a PMA provided inaccurate data as to the number of full actuator fills completed. To correct this problem a counter with pressure thresholds and whose inputs were the PMA pressure states was utilized. This type counter is shown in Figure 29.

The input to this state diagram is the pressure state of one of the PMAs in psi. This diagram is basically the same as the previous one except the conditional statements compare the PMA pressure to a set threshold. In this example the threshold is 170 psi,

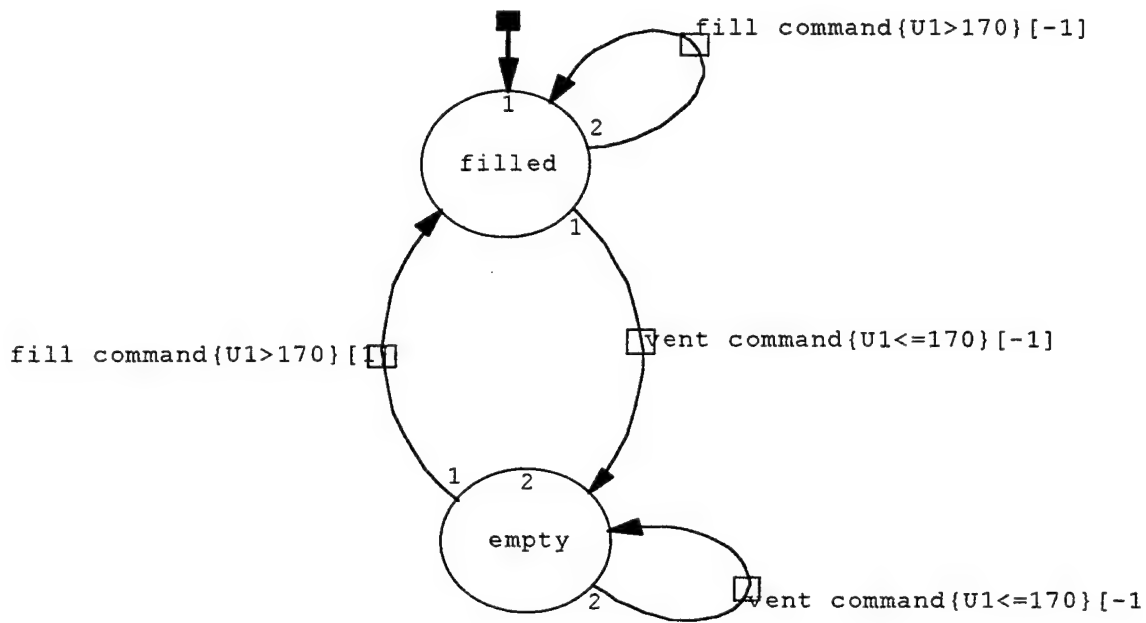


Figure 29. Counter with Thresholds

used with actuators that have a maximum pressure of 175 psi. The assumption here is that if the pressure state of one of the PMAs is above 170 psi, the PMA is considered filled. If this pressure starts to drop below the 170 psi threshold, the PMA is being vented. If after being vented the pressure then begins to rise above the threshold, the PMA is being filled, and this fill is counted by the state diagram. This cycle repeats and the fills are recursively counted through a data store.

This threshold counter can have problems as shown in Figure 30. This plot compares different methods of counting actuations. It also has plots of actuator fill time

changing with time and remaining reservoir pressure changing with time during the simulation. Figure 31 shows the state of the PMAs during this same simulation, and the control logic used.

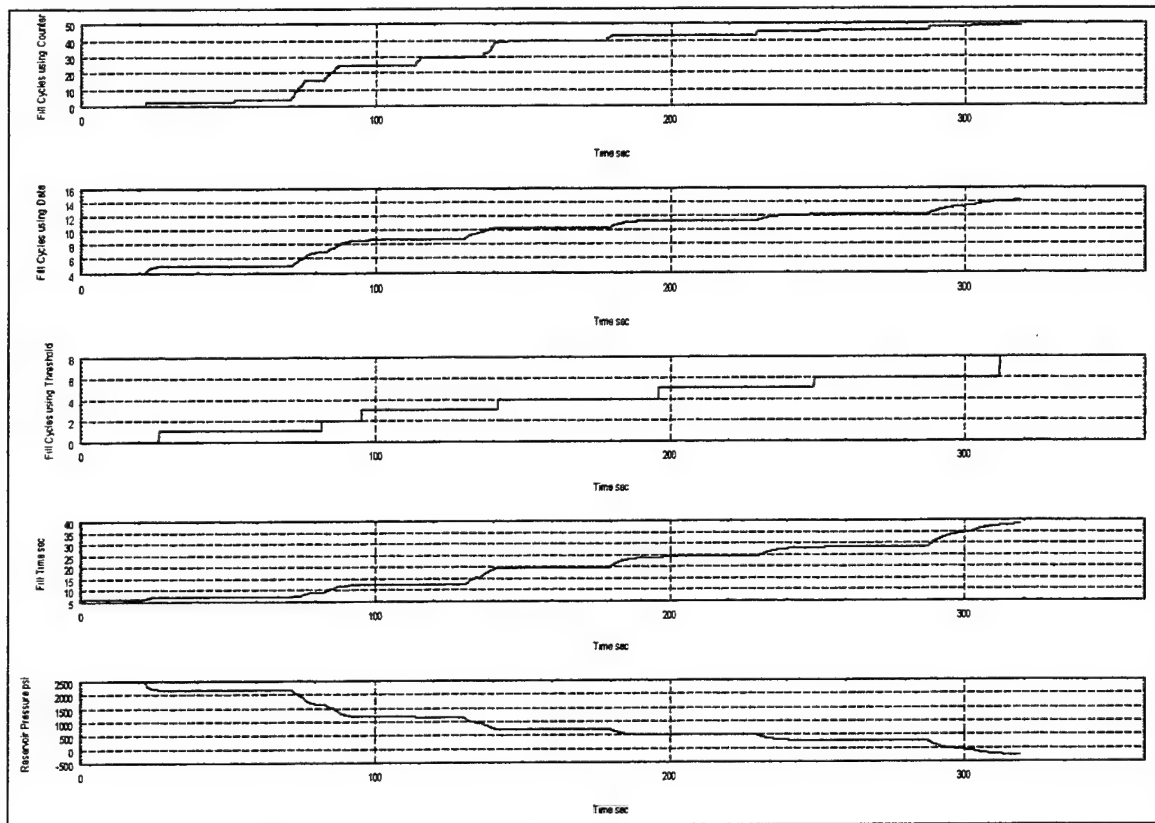


Figure 30. Actuator Data from Simulation

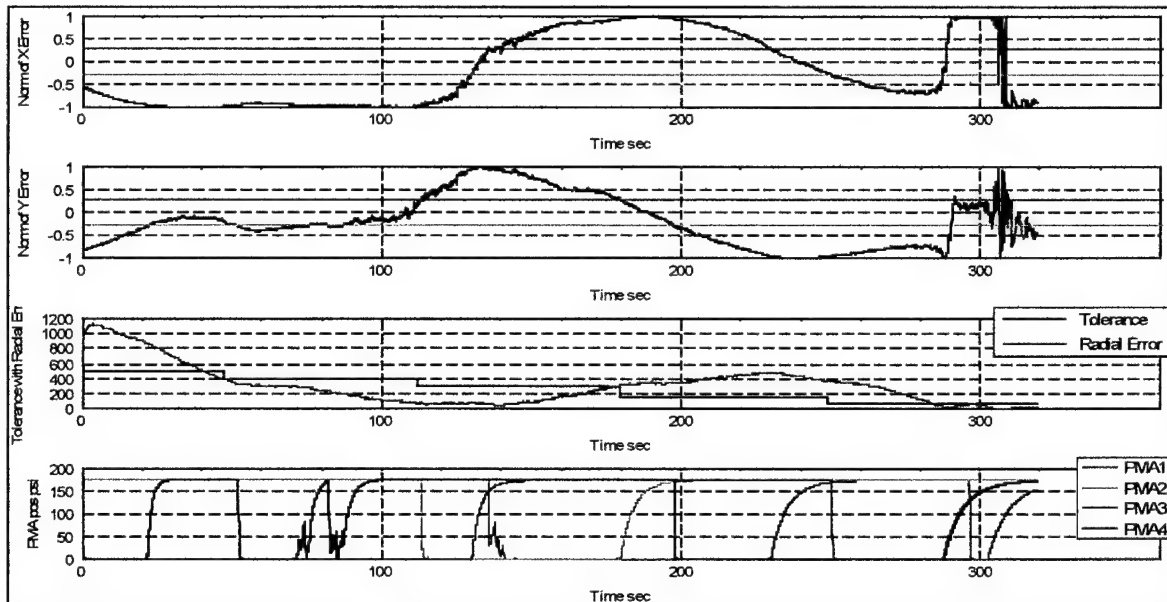


Figure 31. Control Logic for Actuator Data Simulation

The first plot in Fig. 30 shows the number of fill cycles completed with time using the on/off counter. This method drastically overestimates the number of actuator fills. While there are approximately 8 fills of the actuator (not including the initial four actuator fills) seen in Fig. 31, this method calculates this number to be around 50. This is because of the instability of the commands to fill or vent an actuator around the “dead zones”, as seen in Fig. 31.

The second plot in Fig. 30 shows the calculation of actuator fills using data extrapolation. This plot INCLUDES the first four fills upon the parachute release. This method calculates the fill number at 14 (subtracting out the first 4 fills this would be 10), which overestimates somewhat. This is because this data is extrapolated from remaining reservoir pressure, taking into account all the half-fills, quarter-fills, etc. as seen in Fig. 31.

The third plot in Fig. 30 shows calculation of actuator fills using the threshold counter, which only counts a fill if it reaches 170 psi. This method estimates the fill number at 8, which is probably the most accurate number of total actuations. However, this method has some drawbacks in that it does not count the half-fills and quarter-fills that do not make it to 170 psi. Probably a better measure of PMA fuel usage is not number of actuations, but remaining reservoir pressure. In the last plot in Fig. 30, the reservoir pressure for this simulation ran to less than 0 psi, meaning the tank ran out of gas. Future simulations should note the remaining reservoir pressure after parachute landing and use this value as a specification for actuator design. As a side note, the fourth plot in Fig. 30 shows the fill time of the actuators changing with simulation time.

III. DATA COLLECTION

A significant amount of data was collected for this project, mostly for modeling purposes, but also to assess the affect of certain behavior in actual parachute and PMA dynamics. Two aspects of the project had the greatest sense of intangibility: the wind prediction process and the effect of this process on the parachute, and the characterization of the PMAs. This section presents the collection of data for these two critical aspects of the simulation in detail.

A. ACTUATORS

The PMAs are braided fiber tubes with neoprene inner sleeves that can be pressurized or vented, as discussed previously. This allows for the parachute and actuators to be packed easily. A pressurization of an actuator causes a decrease in PMA length and a vent causes the PMA to return to its initial length. Figure 32 shows a picture of the PMAs.

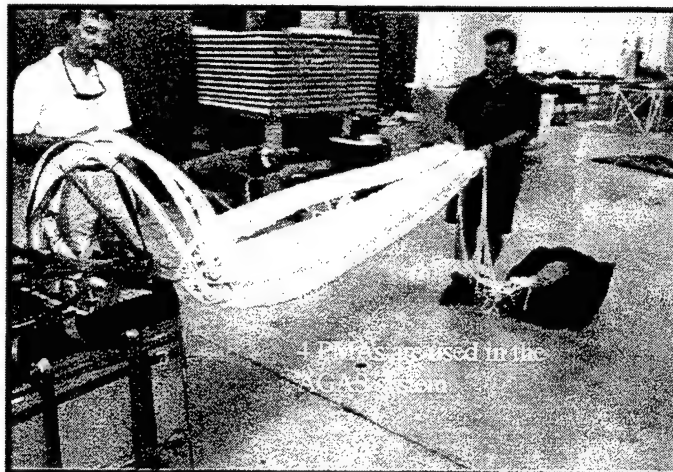


Figure 32. Four Pneumatic Muscle Actuators (PMAs) From Ref. [8]

One particularly crucial area of study was an experiment to attempt to narrow down the exact amount of drive the control actuation of a PMA provided. One idea was to measure the change in length of a particular riser from its nominal length when a PMA was filled to a certain pressure. Figure 33 shows a plot of the data collected from one of these experiments.

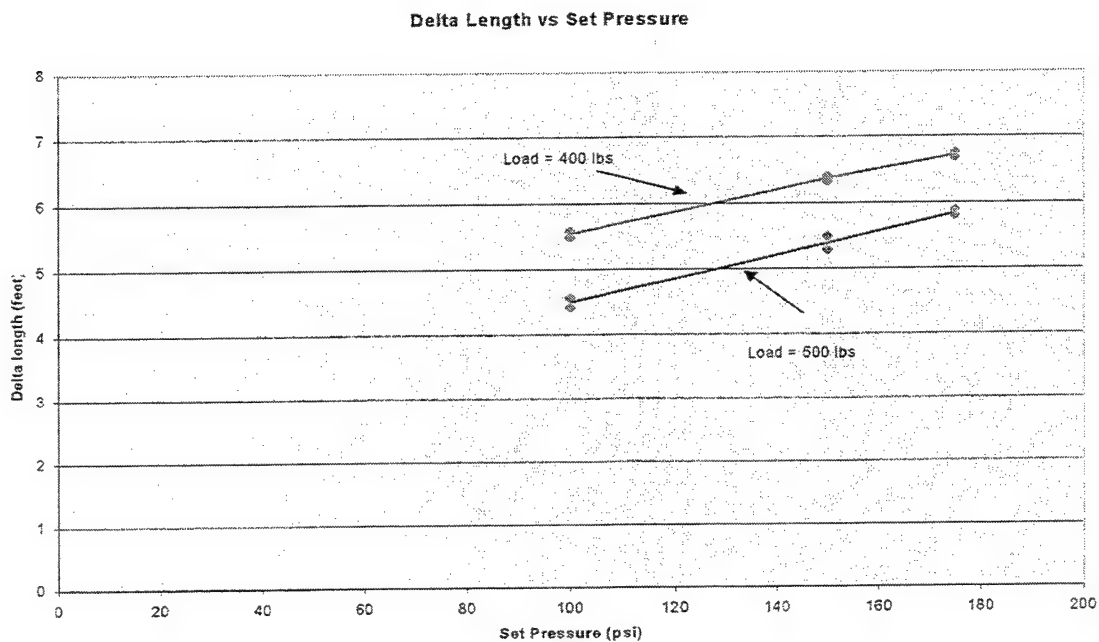


Figure 33. Change in Length of Riser vs. PMA Set Pressure

In this experiment, a certain pressure was set on an actuator and the change in length of the riser attached to the PMA was measured. The PMA also had either 400 or 500 pounds attached to it, as shown by the two plots. At 0 psi, the PMA is at its nominal length, so the delta length is equal to zero at that point. The remaining data was interpolated from 100 psi to 0 psi. It was also assumed that there was a linear

relationship between a change in length of the parachute riser and the force in pounds that this length change provided. In other words, at a full throw of the riser (maximum length change at the maximum actuator pressure) the force provided was zero. At the actuator nominal length (its length when vented), the force provided was either 900 or 317 pounds, depending on how many control inputs were issued at that time. The rest of the data was linearly interpolated. This force based on riser length change is implemented in the parachute model in the block "Force Rotation with Length Change". This assumption is probably inaccurate, but until wind tunnel or actual experiments are done on parachutes with changing riser lengths the assumption stands.

Vertigo, Inc. ran several tests on its actuators to characterize their behavior. They found that the opening and closing of the fill valve took approximately 100 ms, but the filling time took progressively longer with decreasing reservoir pressure. The vent valve opening and closing time also took approximately 100 ms, but the venting process took a longer time, a constant 1.8 seconds. They also ran tests and took data to characterize the

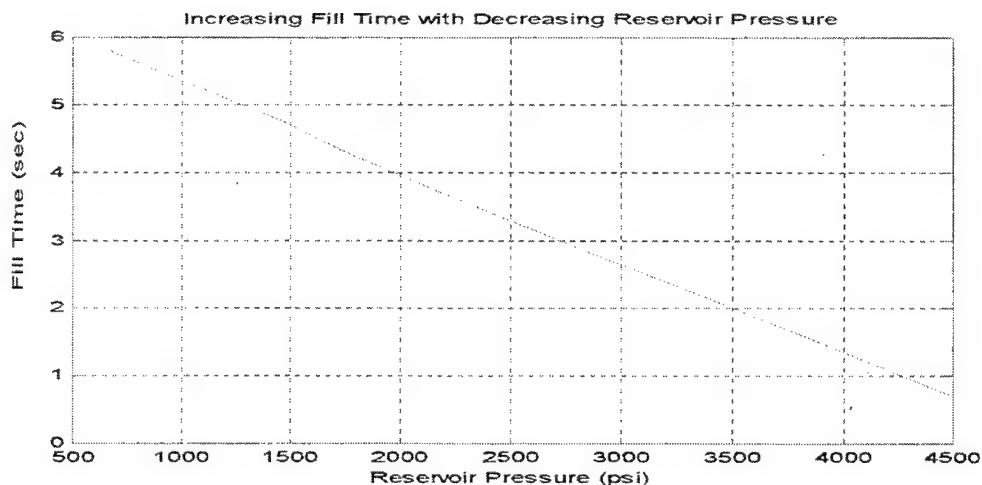


Figure 34. Increasing Fill Time vs. Reservoir Pressure (Vertigo Test) From Ref. [14]

increasing fill time of the PMAs with decreasing reservoir pressure. The maximum pressure of the actuators during these tests was 150 psi. They found this relationship to be largely linear, as shown in Figure 34.

Yuma Proving Grounds (YPG) ran these same tests with actuators acquired from Vertigo and obtained differing results. They filled the actuators to 100, 150, and 175 psi with 4500 psi tanks and a 500 lb load for as many cycles as possible and found that the fill times increased but the relationship was other than linear. Figure 35 summarizes these results.

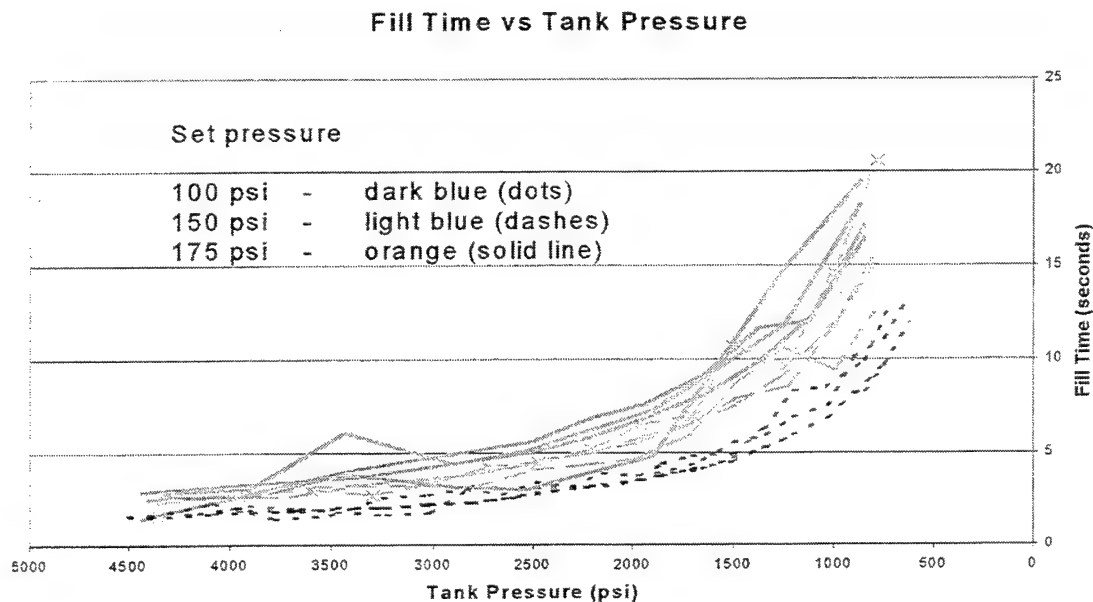


Figure 35. Fill Time vs. Tank Pressure (YPG Test)

Figure 36 presents the data in a different format with fill time and reservoir pressure changing as a function of fill cycle number. Two different methods were used to

calculate the fill time. The first method used pressure transducer data that was digitized and recorded at a 10 Hz rate. This method measured the elapsed time from first detection of increased pressure to the first measurement that crossed the set pressure value. The other method used recorded video to measure the elapsed time from first motion to end of motion of the actuator. The fill times from this method are in general longer than those calculated from pressure data.

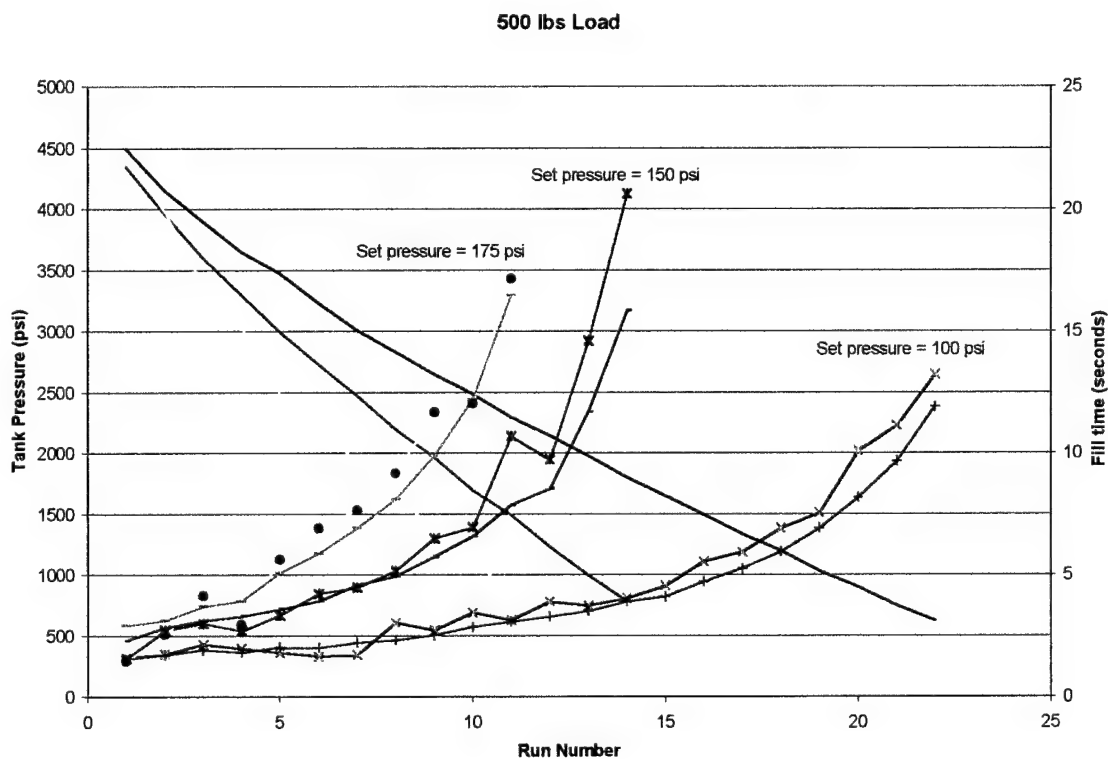


Figure 36. Fill Time and Tank Pressure vs. Fill Cycle Number

This figure also shows that at higher actuator fill pressures a much lower number of fill cycles is allowed.

YPG also found that the time to deflate a PMA did not exactly take the 1.8 seconds tested previously. The chart in Table 1 shows that the average deflate time for each of the set pressures was approximately 0.78 seconds. The column for fill cycle number has the numbers for each actuator fill for each of the set pressures, and the column just to the right has the deflation times for each run number for each of the set pressures.

Fill Cycle Number	100 psi	Fill Cycle Number	150 psi	Fill Cycle Number	175 psi
1	0.934	1	0.801	1	0.767
2	0.868	2	0.9	2	0.768
3	1.268	3	1.234	3	0.734
4	0.701	4	0.868	4	0.8
5	0.767	5	0.867	5	0.767
6	0.734	6	0.7	6	0.734
7	0.7	7	1.635	7	0.801
8	0.767	8	0.701	8	0.768
9	0.668	9	0.735	9	0.801
10	0.667	10	0.741	10	0.834
11	0.701	11	0.767	11	0.768
12	0.7	12	0.734		
13	0.734	13	0.767		
14	0.767	14	0.767		
15	0.7				
16	0.768				
17	0.801				
18	0.801				
19	0.735				
20	0.7				
21	0.734				
22	0.868				

Table 1. Table of Deflation Times

From the data on remaining tank pressure for each fill number, the change in tank pressure for a particular reservoir pressure could be calculated by just taking the next

point's reservoir pressure and subtracting the current reservoir pressure. This data is shown in plot form in Figure 37. The more straight lines are remaining reservoir pressure. The less stable lines are change in tank pressure. The data collected by YPG on remaining reservoir pressure, fill time vs. reservoir pressure, change in tank pressure vs. reservoir pressure, and average deflation time for the 175 psi actuators was used in simulation.

Several flight tests were run with actuators installed on the parachute and controlled from the ground in an attempt to characterize the force coefficients of the actuators. As of this date, the data collected from these flight tests seemed to have had problems or were inconclusive. An idea for an experiment to determine the drive induced by the actuators is to have one actuator vented throughout the entire parachute

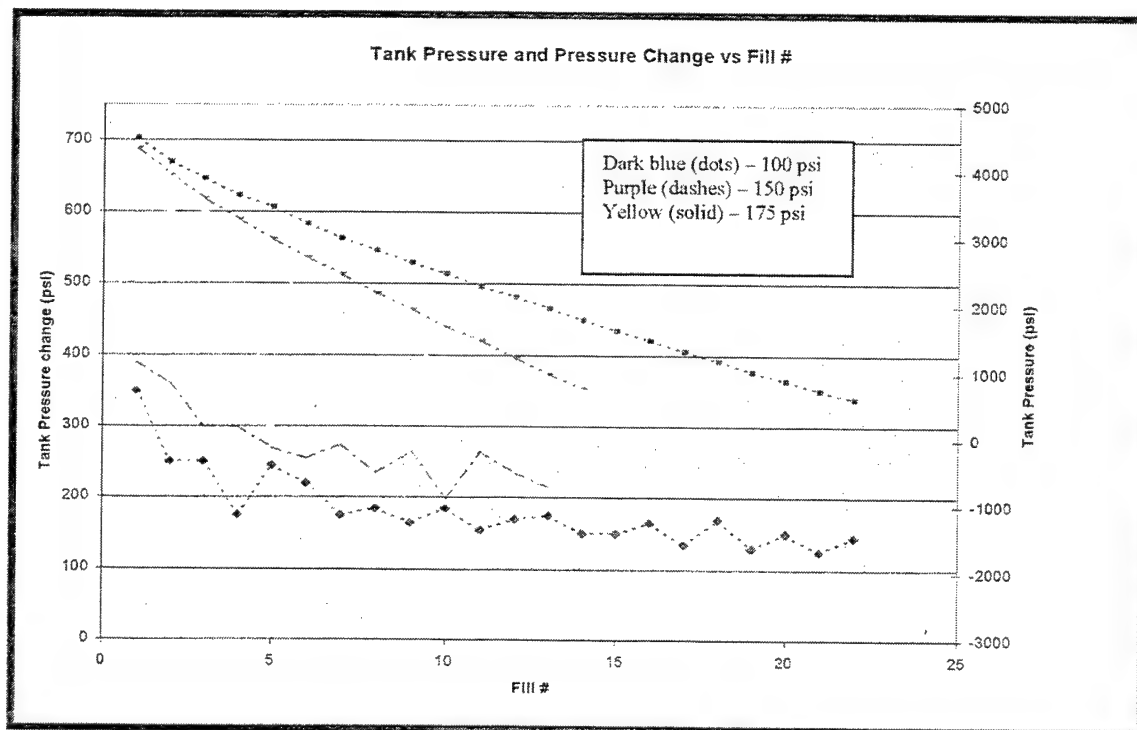


Figure 37. Change in Tank Pressure vs. Fill Number

fall and to measure the glide ratio of the parachute on this fall. Another experiment would be to have two actuators vented during the entire fall of the G-12 and compare the two glide ratios. Force coefficients can then be determined from this data as before.

B. WINDS

Wind data profiles had been collected previously, but for studies on the effect of progressively older wind predictions new data had to be collected. Wind information was gathered from the YPG "Tower M" drop zone using eleven Radiosonde Wind Measuring System (RAWIN) balloons released at one-hour intervals throughout the day on 7 March 2000. The first balloon was launched at 0600 and the last balloon was launched at 1600. Wind data could not be collected at a faster rate because of the limitations of the RAWIN system. The entire process of launching the balloon and collecting and processing the data takes approximately one hour. The magnitude and direction of the wind measured by these balloons is shown in Figures 38 and 39.

In Figure 38, the solid red line is the magnitude of the wind at altitudes from 25,000 ft. down to the ground for the balloon launched at 0600. The rest of the thinner colored lines are the magnitudes of the winds at hourly balloon launches after the first launch. For the altitude range of interest, which was zero to 10,000 ft., the horizontal wind velocity changed by up to approximately 20 ft/sec during the time span of this experiment.

In Figure 39, once again the balloon launched at 0600 is shown in the red heavy line. The others are the thinner colored lines. For the altitude range of interest the wind

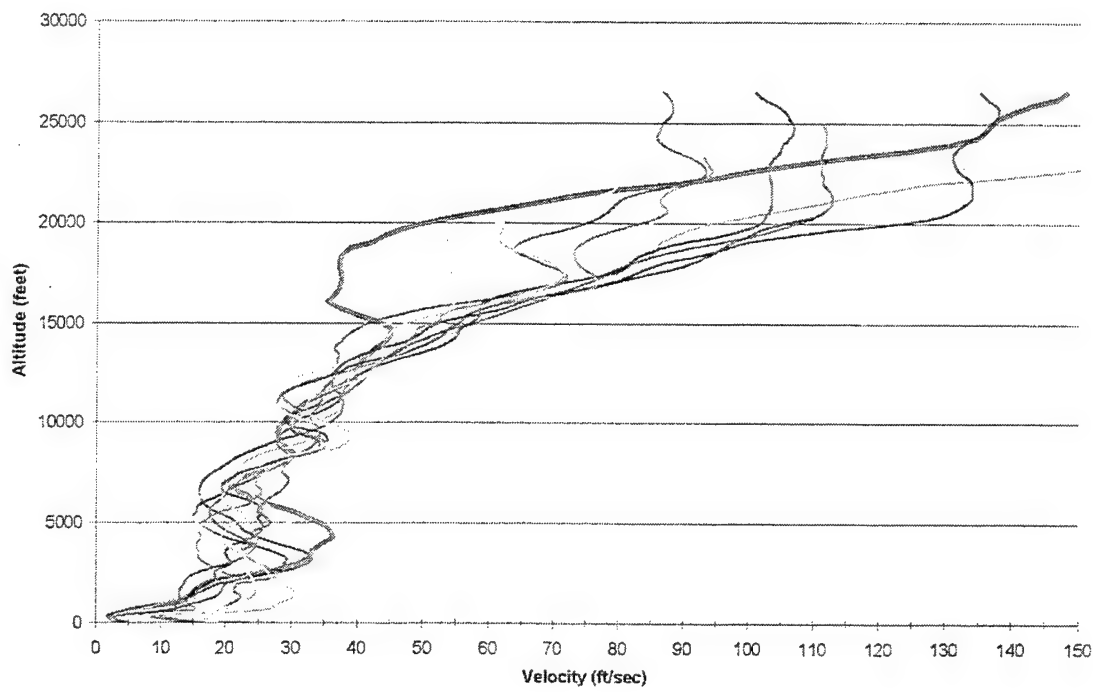


Figure 38. Magnitude of Measured Wind

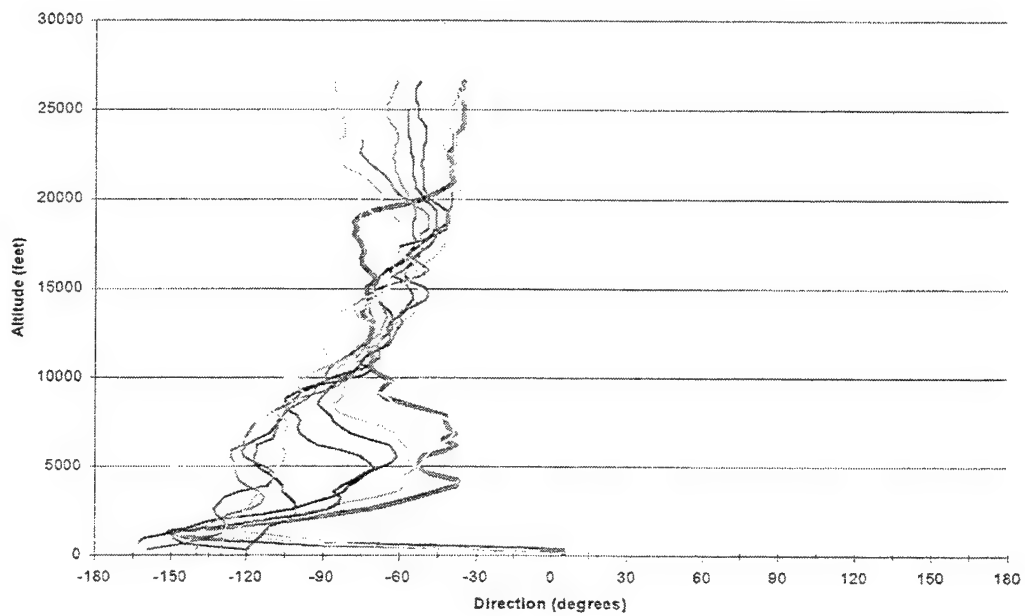


Figure 39. Measured Wind Direction

direction changed by as much as approximately 180 degrees during the time span of this experiment. The wind direction changes become especially drastic at lower altitudes. This plot also shows that as balloon launches were made during the day, the winds changed from being more southeasterly winds (meaning coming FROM the southeast or pointed 000 to -090 in the plot) to being more northeasterly winds (pointed -090 to -180 in the plot).

A simulation was run in which non-controlled parachutes (CARP simulations) were dropped from the same point (0 x position, 0 y position, -9500 ft in NED coordinates) and subjected to the winds measured at each hour. Figure 40 shows a plot of these different trajectories.

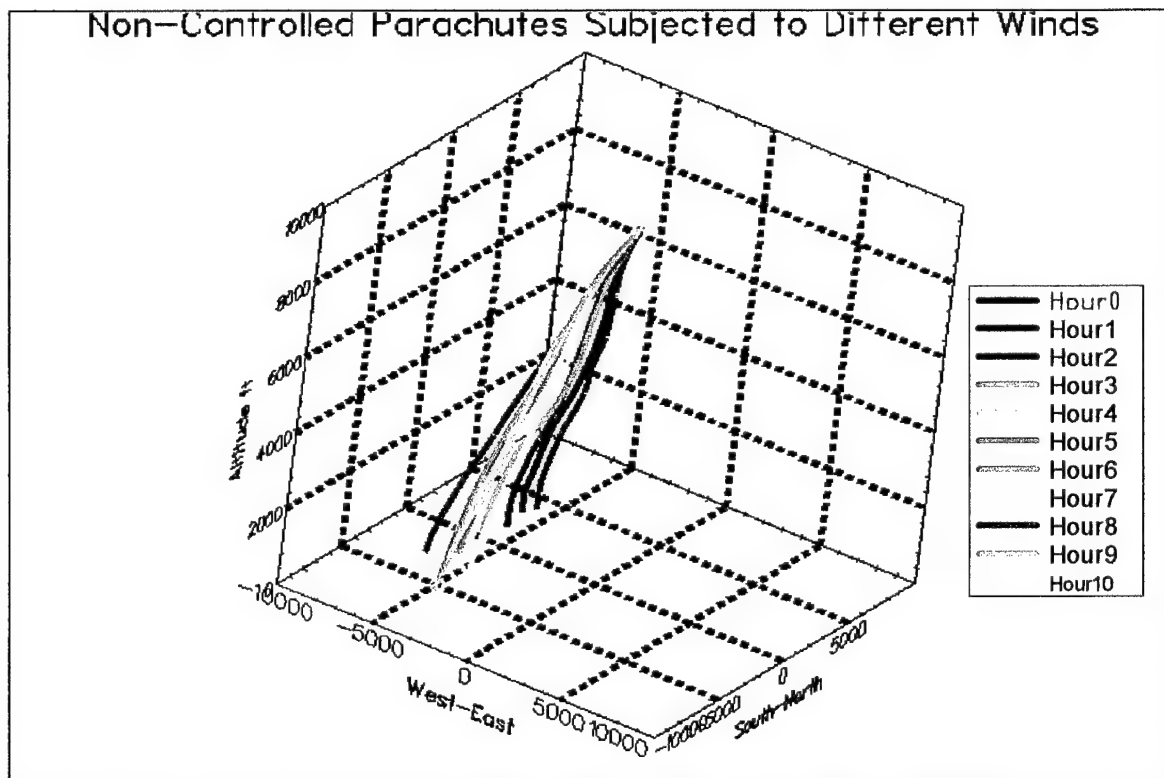


Figure 40. Non-Controlled Parachutes Subjected to Hourly Winds

The plot shows that at Hour 0 (the red line most toward the right side of the graph), the wind forced the parachute in a southwest direction. The plot of wind direction shows a huge direction change in the southwest direction toward the lower altitudes, so this is consistent. However, as the drops go on during the day, the impact points become more south of this first impact point. This is consistent with the wind direction data showing the winds shifting from pushing the parachute towards the northwest to pushing the parachute to the southwest. The plot also shows that such wind changes cause a difference in impact points of the CARPs of more than 5000 ft. at 10-hour-old wind predictions.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SIMULATION

A. PROCESS

Two sets of simulations were run as trade-off studies to assess the affect of two crucial aspects of the parachute control design: (1) a simulation comparing two control strategies at random wind predictions and offsets from the ideal drop point, and (2) a comparison of simulations using different actuator models to assess the affect of longer fill times. Programs were developed using XMATH/MathScript[®] computing language to run these simulations and are included in Appendix A. The SystemBuild[®] model described in previous chapters was utilized as a model of the parachute, sensors, actuators, and control system.

B. RESULTS

A first assessment comparing the two control strategies was run in a file called "agesims". In this simulation, 175 psi actuators were used. The predicted wind file for the CARP trajectory was the first hour prediction from 0600 7 March 2000. Selective availability was off (a more accurate GPS) and there was no offset from the ideal release point for the controlled parachutes. The target point for all the simulations was the point at zero altitude and 0 x and 0 y position. A strategy that sought the predicted trajectory from CARP at all times (the so-called "trajectory-seeking" control strategy) was run and a strategy that sought the target position at all altitude stations ("target-seeking") was run. Figures 41 and 42 are 3-D plots of these runs. Table 2 compares the impact errors from the target for each of the succeeding runs using the later wind file as an actual wind.

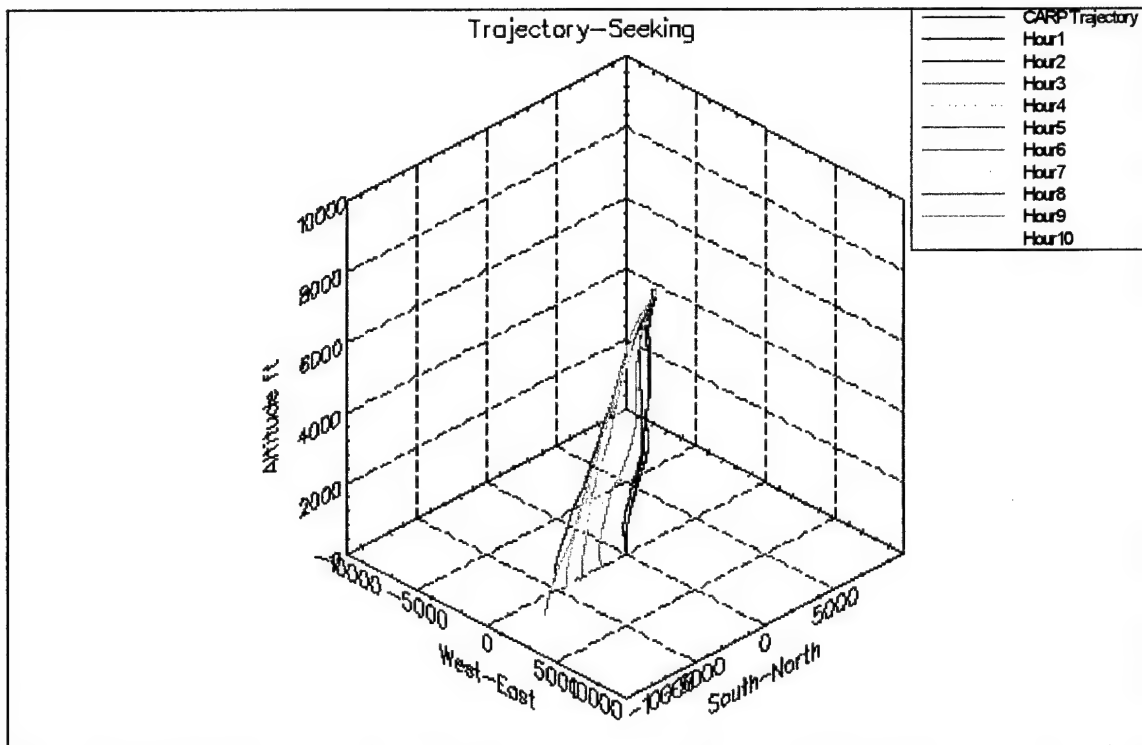


Figure 41. Trajectory-Seeking Control Strategy Age-of-Wind Comparison

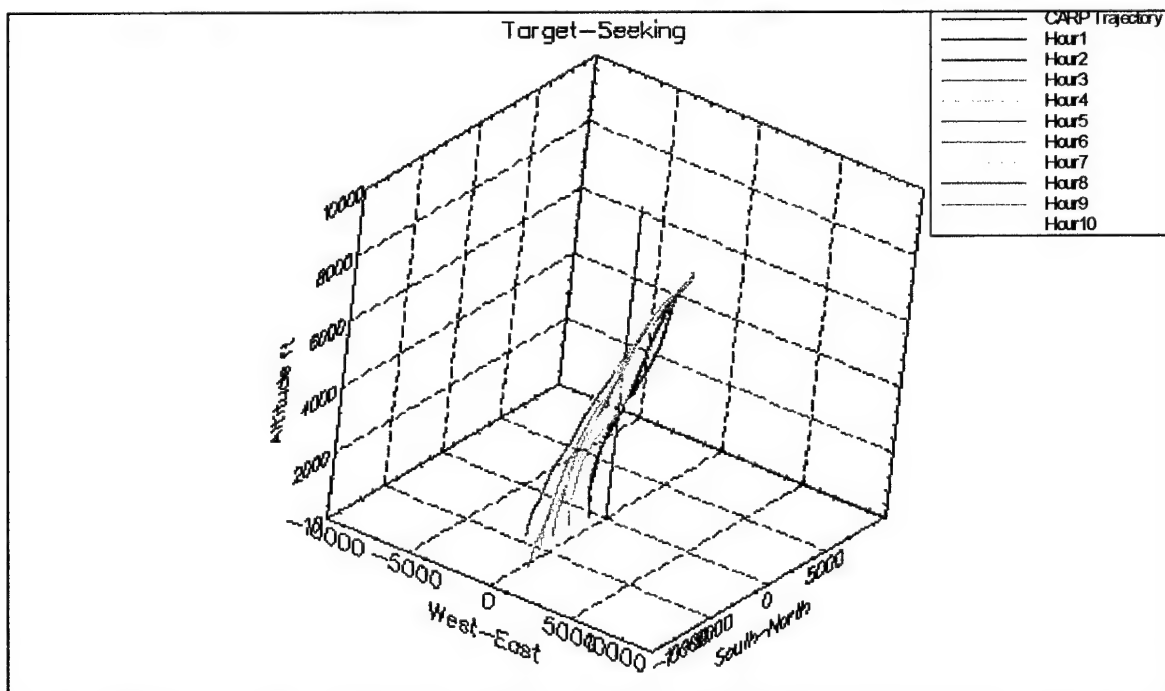


Figure 42. Target-Seeking Control Strategy Age-of-Wind Comparison

Time of Drops	Trajectory-Seeking Error (ft)	Target-Seeking Error (ft)
7:00 AM	13.5058	860.934
8:00 AM	36.1824	881.149
9:00 AM	1744.5	2266.81
10:00 AM	2323.83	2790.61
11:00 AM	3383	3674.62
12:00 PM	4721.48	4883.06
1:00 PM	4542.89	4695.46
2:00 PM	4524.03	4943.23
3:00 PM	7106.98	7178.22
4:00 PM	6604.91	6528.56

Table 2. Control Errors for Simulation

The predicted trajectories in Figures 41 and 42 are represented by red lines. In the target-seeking strategy, this is simply a line extending vertically upward from the target point. The plots show that since the early wind files did not change much from the predicted wind file, the errors for those drops 2 or less hours from the wind prediction were very good. The wind direction changed very much afterwards, however, and the parachute was not able to overcome the bad wind estimate. Since the wind changed to blow more towards the south, most of the impact points occurred south of the desired impact point, as shown in the 3-D plots. The parachute was following a trajectory that did not account for this sudden wind change. Also, the wind changes were up to 20 ft/sec in the opposite direction (south) the parachute control system thought the wind was going to blow (north). At a maximum 0.4 glide ratio and constant descent rate of 25 feet/sec, the greatest change in velocity the drive of the actuators can overcome is 10 ft/sec. The parachute cannot possibly overcome a wind of this magnitude with the bad wind prediction. The data in most cases supports the fact that for this simulation an older wind

prediction accounts for larger impact errors. For the comparison of the two control strategies, when the wind prediction was good (i.e. at 7:00 AM and 8:00 AM) the trajectory-seeking strategy worked better. However, in one case where the wind prediction was bad (i.e. 4:00 PM), the target-seeking strategy was slightly better. This test merits more investigation and a greater number of simulations with different combinations of wind profiles and offsets from the desired drop point.

For the next set of simulations, this exact study was repeated but with random offsets from the ideal drop point and a random selection of wind profiles from the 11 already discussed. This simulation was run through a script file called "runmanysimsnewwind". In this file, each of the wind files from 0700 to 1600 were used as actual winds for the parachute. They were set based on the number of simulations desired. For each of the actual winds, a uniform randomly distributed wind file for the predicted winds was chosen from the wind files PREVIOUS to the actual wind file. So for the actual wind file at 0700 the only choice for a predicted wind was the one at 0600, etc. Offsets were also chosen randomly. The offsets in x and y position from the ideal drop point were normally distributed about a mean of zero. The maximum offset, determined from the "area of attraction" simulation to be approximately 2500 feet radially or 1767.8 feet in x and y, was set at four standard deviations. This corresponds to the pilot missing the ideal drop point by the maximum offset about 1 in 1000 times. The "trajectory-seeking" and "target-seeking" control algorithms were both run for the same offset and winds and several sources of data collected on the two through a called

script file called "runsim". Once again the target position was the (0,0) point on the ground and selective availability was off.

1000 simulations were desired for this study. Because of problems with the personal computers running the simulations, only 437 simulations were able to be run, and of these 437 simulations most of them were biased toward more recent (less than 2 hours old) wind information. Figure 43 is a histogram of the age of the winds used in the

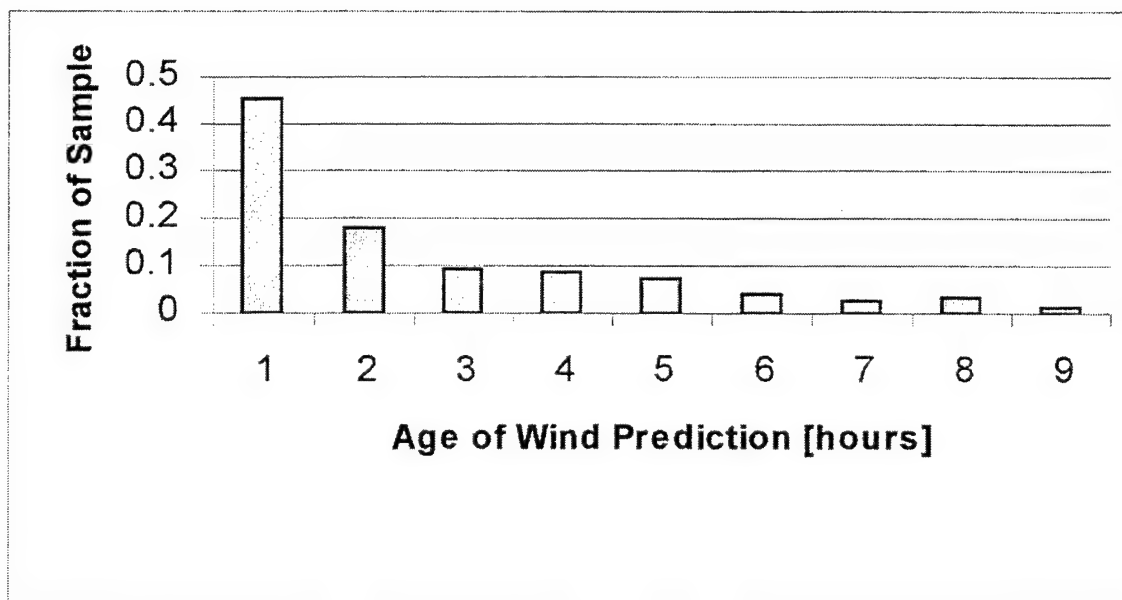


Figure 43. Distribution of Ages of Wind Predictions (437 samples)

simulations. Figures 44 and 45 are polar plots for the "trajectory-seeking" and "target-seeking" control strategies, respectively.

Each of the circular rings in these polar plots represents 2,000 ft. The black stars in Figures 44 and 45 are the ideal drop points. They are all east of the target point, which

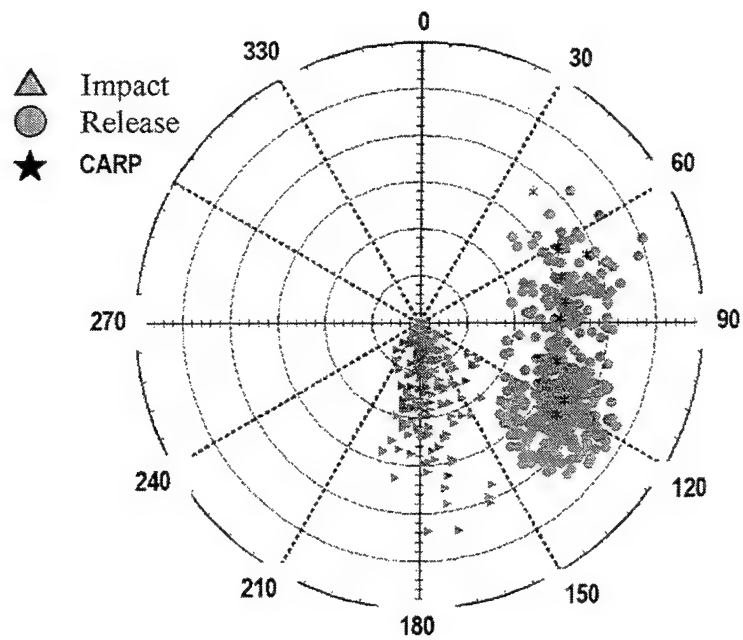


Figure 44. Trajectory-Seek Impact and Release Points

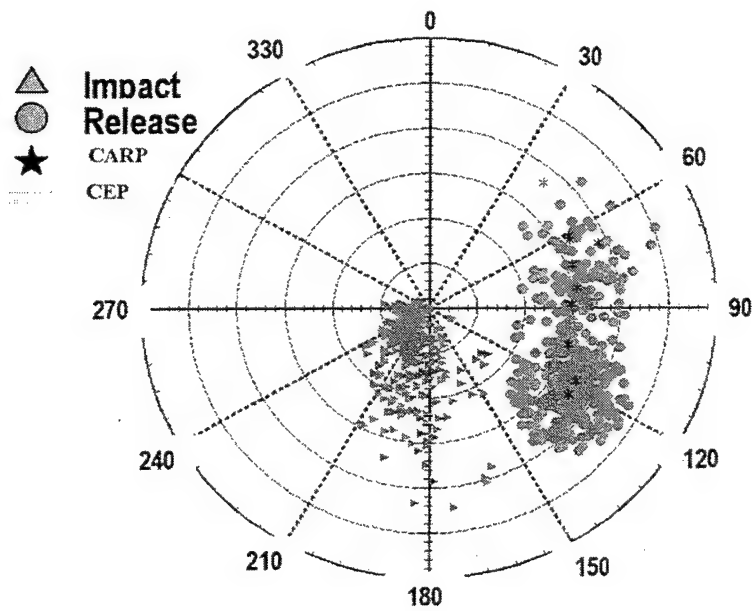


Figure 45. Target-Seek Impact and Release Points

is consistent with the fact that all the winds for the most part blow toward the west. The red dots are the actual release points scattered around the ideal drop points. The blue triangles are where the controlled parachutes landed. Most of the drops landed south of the target zone, which is consistent with the wind changing from blowing toward the north to toward the south.

It may seem that many of the controlled parachutes fell outside the CEP, or ideal circular area around the target of 100 meters. However, a closer look shows surprising information. Figure 46 and 47 show the two zoomed in on the CEP, with only the impact points of the controlled parachutes plotted.

These figures show that the density of impact points within the CEP was actually high. Figure 48 shows the statistics of the control errors (as well as errors for non-controlled parachutes subjected to the same winds). It is assumed that this accuracy is due to the majority of wind estimates being 2 or less hours old. The domain of the histogram is in meters, with 100 m CEP being the goal of the parachute drops. For this set of simulations, over 50% of the drops using "trajectory-seeking" reached this goal.

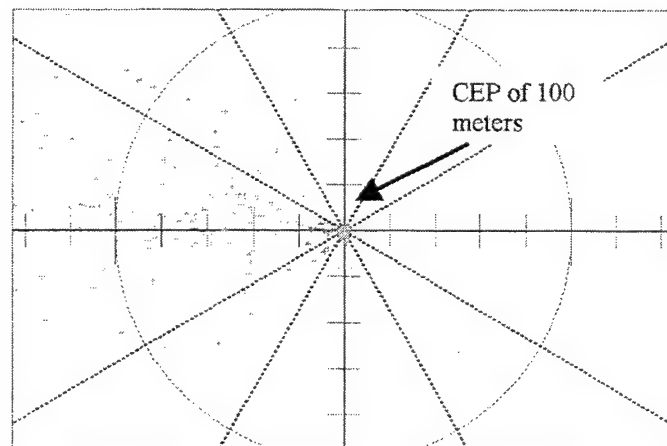


Figure 46. Trajectory-Seek Impact Points Zoomed In

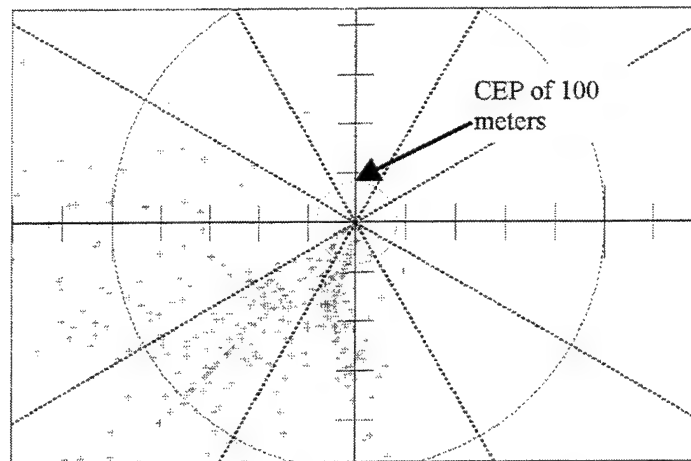


Figure 47. Target-Seek Impact Points Zoomed In

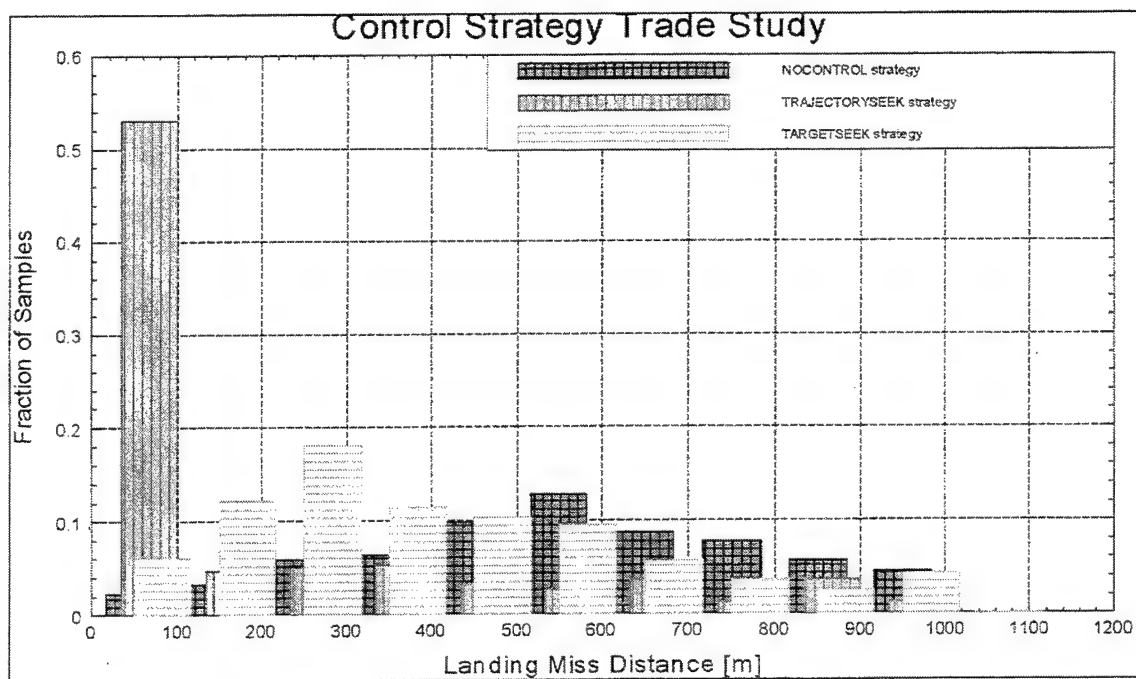


Figure 48. Distribution of Parachute Landing Miss Distances

For a worse case scenario, the same study was conducted but the wind from time 0800 was chosen as the predicted wind, and random ACTUAL winds were chosen from the wind files after this predicted wind. Thirty simulations were run and their trajectories plotted using a script file called "runmanysimsandplot0". These simulations provided worse impact errors, presumably because the wind from these hours had more drastic changes. Also, there were no computer problems during the simulation, so there was no bias in the results as before. Figure 49 shows plots of the trajectories zoomed in on the impact points. These plots show more drastic errors for both the "target-seeking" and "trajectory-seeking" control strategies.

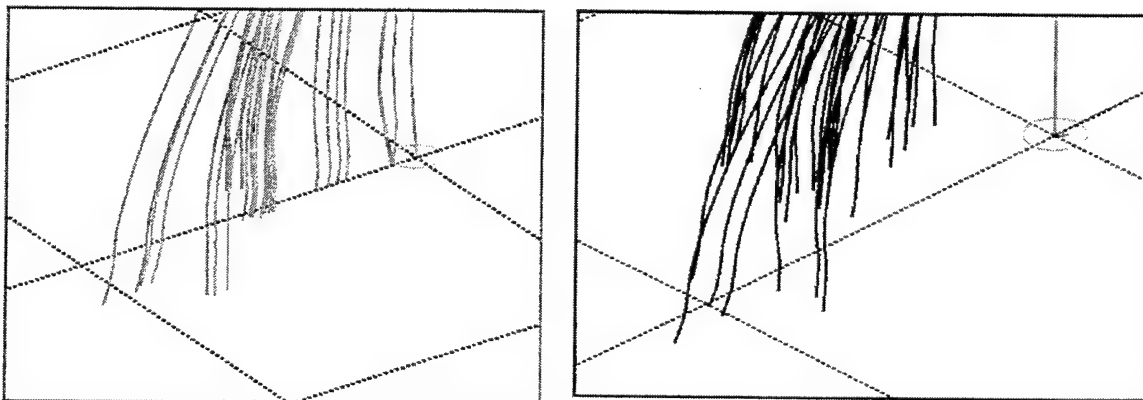


Figure 49. Thirty Trajectories for Trajectory-Seek and Target-Seek Strategies

Table 3 is a comparison of statistical data for the two control strategies, including data on number of actuator fills for the study in which the wind predictions were biased to the earlier winds of the day.

			Control Error [ft]			Number of Fills		
			Mean	Std. Dev.	Max.	Mean	Std. Dev.	Max.
NOCONTROL			3021.6	2147	11246.3			
TRAJECTORYSEEK			1264.8	1798.8	8823.4	11.1	1.3	16
TARGETSEEK			1879.6	1598.9	8870.5	10.3	1.5	13
Notes:								
437 trials were performed								
- number of control actuations performed includes four actuations for the initial filling of the PMAs								
- wind predictions ranged from one to ten hours old, but trials were skewed towards having more								
Current wind predictions								
- actuators had unlimited fuel supply, but fill time increased with increasing number of actuations								

Table 3. Statistical Data on Control Strategy Comparison

This data only confirms what was already expected. The “trajectory-seeking” strategy performed better with the more precise wind estimations. The number of fills is a design benchmark for the actuators. These fills were counted using the threshold counter described in the section on actuators. It is assumed that the “trajectory-seeking” strategy completes more fills on average because it does more error correction around a moving trajectory. The “target-seeking” strategy merely aims at a single target throughout its entire drop.

Figure 35 in the section on data collection showed the experimental data collected by YPG on the PMA fill time changing with decreasing reservoir pressure currently installed on the AGAS (the 175 psi actuators). For performance research related to actuator characteristics, a set of actuator models was created with the baseline model representing the system in its current form. Four additional models were created: two models with better fill time characteristics with decreasing tank pressure, and two models

with worse fill time characteristics. These models were created by simply fitting exponential lines through data points with worse and better fill time characteristics. The baseline model was named “average,” and the other four models were named “best,” “better,” “worse,” and “worst,” respectively. A plot of fill time with respect to tank pressure for these models is shown in Figure 50.

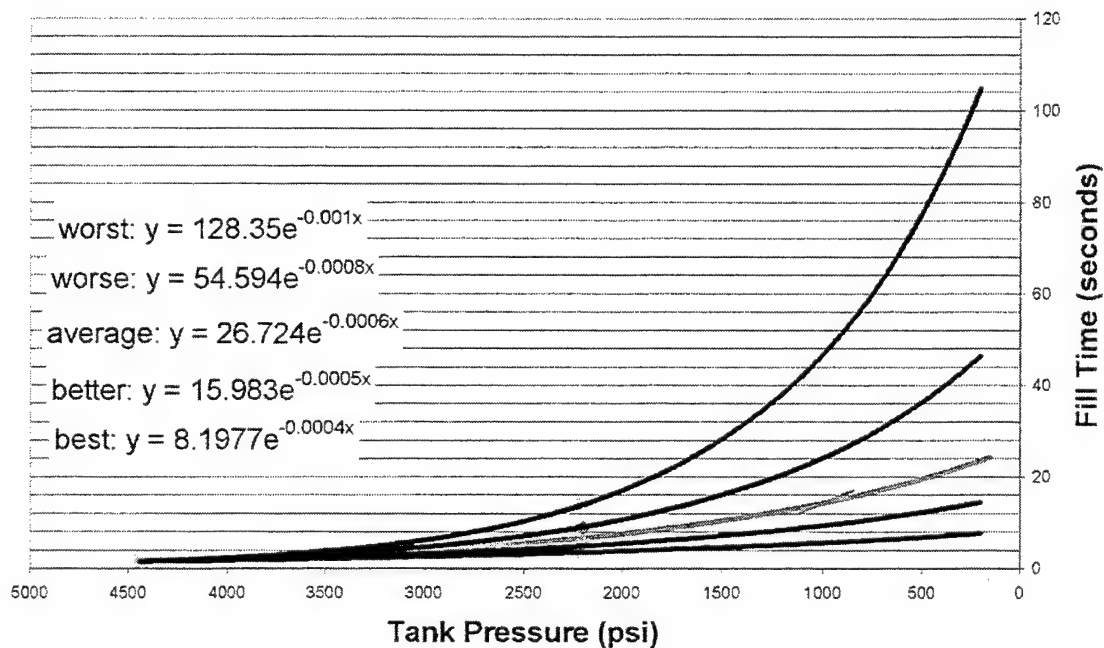


Figure 50. Fill Time vs. Tank Pressure for Five Actuator Models

For this study 500 total simulations were run, 100 for each of the different actuator fill time models. A script file called “runmanyactsims2” set the wind for the predicted trajectory at the 1100 RAWIN balloon launch. The actual wind was set at the 1300 launch. These particular winds were picked because the average amount of time

given for a wind prediction before airdrop is approximately two hours. Also, these two winds were very similar, eliminating another random variable in this actuator study. The offsets from the ideal drop point were once again normally distributed around a mean of zero with the max offset at four standard deviations. In essence, the offset was the only variable aside from small errors in GPS and compass. The file set up 100 iterations, with each iteration simulating the "trajectory-seeking" control strategy for each of the five actuator models. Each of the five simulations used the same offset, wind, and sensor seeds (using SA off GPS). The file called another file named "runactsim2", which ran the simulations and collected data. Table 4 is a statistical analysis of the actuator study.

	Miss Distance [ft]			Number of Fills			Max. PMA Fill Time		
	Mean	σ	Max.	Mean	σ	Max.	Mean	σ	Max.
Best	202.5	20.2	233.8	12.7	2.1	18	8.0	1.3	10.8
Better	199.2	22.4	232.8	11.7	1.4	16	14.2	2.6	19.4
Average	195.4	21.9	234.5	11.2	0.9	14	22.2	4.1	30.5
Worse	195.0	22.7	234.5	10.5	1.2	14	39.2	7.7	52.8
Worst	203.0	22.8	274.5	10.6	1.2	13	83.0	18.4	112.3

Table 4. Actuator Characteristic Research Simulation Results

From these results, two observations were made. First, it seems that the changing fill time characteristics among the different models seemed to have little effect on the miss distance achieved. Also, the actuator models that had a lower overall PMA fill time used more PMA fills. This observation was thought to be due to the manner in which the fills were counted. Remember, with the threshold counter, one complete fill was counted when the PMA filled to slightly below its maximum pressure (170 psi). For those actuator models where the fill times were very long, it is possible that, after a PMA is

commanded to fill, the same PMA was commanded to deflate again before its pressure reached the threshold for a fill to be counted. Such inconsistency in counting actuator fills probably means that a more accurate method of providing a benchmark for actuator design is measuring remaining tank pressure upon impact. Also, the fact that the winds were set to a good prediction (2 hours old) may not be a good way to assess the affect of higher or lower fill times. A possible new experiment would be to have the winds randomized also, and compare age of wind and actuator model used to control error.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The simplistic model of the G-12 parachute dynamics was successfully transferred over to an XMath/SystemBuild[®] environment for implementation on the AC-104 real-time controller. Actuators were also modeled on the computer, tested, and verified. More research was done to study performance of the controller and possible ways to optimize its response. Additional data collection on wind and actuator behavior provided a basis for continuing research into the characterization of both the wind prediction process and the dynamics of the actuators. Simulation results found that the wind estimation process is the crucial aspect of the entire control scheme. Without a good wind prediction, errors in the control can be great. Simulation also found that changing fill time does not affect the control of the parachute. These results could change with a more complex model and more information on the dynamics of the actuators. The Affordable Guided Airdrop System is definitely a feasible and promising program, but many questions still remain to be solved, and much research into the system remains to be done.

B. RECOMMENDATIONS

The following recommendations are based on observations made during the simulation process and work needed to be done before initial flight tests of the complete system:

1. Investigation into a more complex parachute model. Once a more complex model is implemented into the above simulations, comparisons and a decision as to whether or not a more complex model is necessary can be made.

2. Additional flight tests of the parachute are needed to characterize the parachute's motion for a more complex model analysis, characterization of the drive of the actuators, and effect of wind prognosis.

3. The wind estimation process needs to be refined. It is clear at least from simulation that wind prediction is a major aspect of the entire control process.

4. Obtain more data on the actuators and compare simulation data with actuator test data in order to refine the actuator computer model. Run more simulations with new actuator data and a more complex parachute model to assess the affect of changing fill times with the new model.

5. Run simulations to determine average and maximum reservoir pressure needed vice number of actuations. The counting of number of actuations is a tricky process. Reservoir pressure might be a better measure for redesign of the PMAs.

6. Run simulations testing different control algorithms, including:

- Allowing only one control actuation at a time to take advantage of the better glide ratio; in this case the actuator turned on would be the one in which the line of sight angle to the target is largest, or the most in its operating angle (see Appendix B for code).
- Playing with the tolerance cone to maximize its usage; the tolerance cone would possibly be non-symmetric (non-circular) based on wind data.

- Changing control strategies at a point where one strategy is not providing a closure of the radial error from the target; in this case, the derivative of the error would have to be calculated, and if it were positive then a switch from “trajectory-seeking” to “target-seeking” would be in order.
- A predicted trajectory look-up table that provides as its target NOT the point in space on the predicted trajectory at the same altitude as the actual drop, but rather a point at a lower altitude the parachute could glide to.
- A possible control strategy in which the system uses wind trends to set up the control logic, so as not to be put in a position in which the parachute is controlling against the wind.

7. Test through simulation the effect of having both direction (clockwise and counterclockwise) and greater rotation rates.

8. Additional simulations need to be run with winds from varied climates

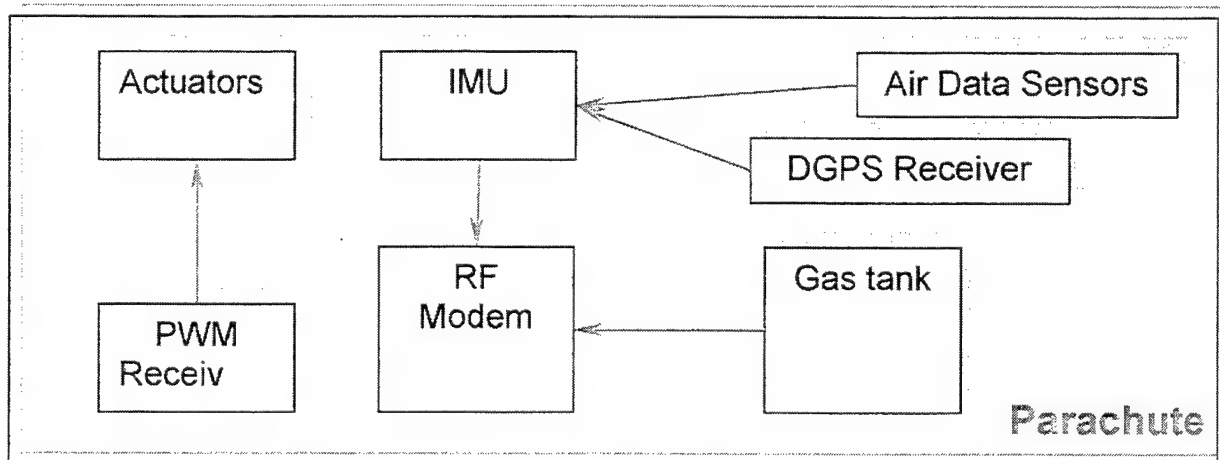


Figure 51. Actuator-in-the-Loop Testing Scheme

and terrains in order to assess the affect of these different winds.

9. Set up an actual PMA on the ground station and begin initial hardware-in-the-loop-testing. An example of such a ground station is shown in Figure 51.

10. Additional analysis is needed into optimal control techniques, possibly the minimization of a cost function describing either minimized fuel usage or minimized time [Ref. 15].

LIST OF REFERENCES

1. "Summary Report: New World Vistas, Air and Space Power for the 21st Century," United States Air Force Science Advisory Board, 1977.
2. Accorsi, M., Benney, R., Leonard, J., and Stein, K., "Structural Modeling of Parachute Dynamics," *AIAA Journal*, Vol. 38, No. 1, January 2000.
3. White, F. and Wolf, D., "A Theory of Three-Dimensional Parachute Dynamic Stability," *Journal of Aircraft*, Vol. 5, No. 1, January-February 1968.
4. Ayres, R. and Tory, C., "Computer Model of a Fully Deployed Parachute," *Journal of Aircraft*, Vol. 14, No. 7, July 1977.
5. Doherr, K. and Saliaris C., "On the Influence of Stochastic and Acceleration Dependent Aerodynamic Forces on the Dynamic Stability of Parachutes," AIAA-81-1941.
6. Wright, B., "Wind-profile Precision Air Delivery System (WindPADS)," Slide Presentation, Natick, MA, May 2000.
7. *MATRIX-X Online Documentation*, Integrated Systems, Incorporated, Sunnyvale, CA, 1999.
8. Dellicker, S., *Low Cost Parachute Guidance, Navigation, and Control*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1999.
9. Kaminer, I., "Lecture Notes, AA4276 Avionics Systems Design," Naval Postgraduate School, Monterey, CA, 1998.
10. Kaminer, I., "Lecture Notes, AA3276 Introduction to Avionics," Naval Postgraduate School, Monterey, CA, 1997.
11. Kaminer, I., "Lecture Notes, AA4342 Control Systems Design," Naval Postgraduate School, Monterey, CA, 1998.
12. "Innovative Affordable Airdrop Technology," Slide Presentation, The Boeing Corporation, December 1999.
13. "AGAS 1999 Overview Briefing Vertigo Subcontract," Slide Presentation, Vertigo, Incorporated, Lake Elsinore, CA, 1999.

14. "AGAS PMA Fill and Vent Characteristics," Vertigo Document 1801-18-0004, Vertigo, Incorporated, Lake Elsinore, CA, April 1999.
15. Boltjanskiy, V., Gamkrelidze, R., Mishenko, E., and Pontrjagin, L., "Mathematical Theory of Optimal Processes," Nayka, Moscow, 1969.

APPENDIX A. SCRIPT FILES USED IN SIMULATION

1. Math script file "attraction.ms"

```
{ batchfile: This batchfile runs the CARP predictor with an exactly  
right wind file (one of the "actual" wind files). It then runs the  
"TOP" simulation from an offset (in x and y) to narrow down the "area  
of attraction"  
}#
```

```
# set wind files--should be the same to find area of attraction
```

```
wind.forecast_alt=wind.Wind0(:,1);  
wind.forecast_x=wind.Wind0(:,3);  
wind.forecast_y=wind.Wind0(:,2);  
wind.forecast_z=wind.Wind0(:,4);
```

```
wind.actual_alt=wind.Wind0(:,1);  
wind.actual_x=wind.Wind0(:,3);  
wind.actual_y=wind.Wind0(:,2);  
wind.actual_z=wind.Wind0(:,4);
```

```
# set actuator data
```

```
pma_max_pressure = 175;  
pma_max_pressures = [175,175,175,175];  
del_p_vs_p=actuators.del_p_vs_p175;  
fill_time_vs_p=actuators.fill_time_vs_p175;  
del_res_per_PMA=actuators.del_res_per_PMA175;  
fills_vs_respress=actuators.fills_vs_respress175;  
deflate_time=actuators.deflate_time175;  
init_reservoir_press=actuators.init_reservoir_press175;
```

```
# sim CARP
```

```
q=sim("CARP", t, {ialg="VKM"});
```

```
# create predicted trajectory
```

```
pred_mat=makematrix(q,{channels});  
predicted_x=pred_mat(:,1);  
predicted_y=pred_mat(:,2);  
predicted_z=pred_mat(:,3);
```

```
delete pred_mat;
```

```
# set GPS random number seeds
```

```
set distribution uniform;  
saseed=round(9999*random(1,9),{up});  
saseed1=saseed(1); saseed2=saseed(2); saseed3=saseed(3); saseed4=saseed(4);  
saseed5=saseed(5); saseed6=saseed(6); saseed7=saseed(7); saseed8=saseed(8);  
saseed9=saseed(9);
```

```
# selective availability on or off (0 is off, 1 is on)
```

```

saon=0; # equal to 0 always now because of President's declaration

# random compass bias between -2 and 2 degrees
set distribution uniform;
compass_bias=4*(random(1,1)-0.5);

# set offset of vehicle trajectory
x_offset=2500; # area of attraction is a circle of radius approximately 2500 ft.
y_offset=0;
linear_position_init = [x_offset; y_offset; drop_alt];

# sim vehicle model
y=sim("TOP", t, {ialg="VKM"});

# sim no control model
p=sim("NOCONTROL", t, {ialg="VKM"});

# calculations
TOP_mat=makematrix(y,{channels});
controlled_x=TOP_mat(:,1);
controlled_y=TOP_mat(:,2);
final_con_x=controlled_x(length(controlled_x));
final_con_y=controlled_y(length(controlled_y));
final_CARP_x=predicted_x(length(predicted_x));
final_CARP_y=predicted_y(length(predicted_y));

control_error=((final_con_x-final_CARP_x)**2+(final_con_y-final_CARP_y)**2)**0.5;
display("control_error = "); display(control_error);

# plots
plots.att_graph2D=plot(y(1,1), y(2,1), {x_min = -7500, x_max = 7500, y_min = ...
-7500, y_max = 7500, title = "2D Trajectory", line_color = "blue"}})?
plots.att_graph2D=plot(p(1,1), p(2,1), {line_color = "green", keep})?
plots.att_graph2D=plot(q(1,1), q(2,1), {line_color = "red", keep, ...
legend = ["Vehicle Trajectory", "No Control Trajectory", "CARP Trajectory"], ...
xlab="X", ylab="Y"}})?

plots.att_graph3D=plot(y(1,1), y(2,1), -y(3,1), {x_min = -7500, x_max = 7500, y_min = ...
-7500, y_max = 7500, z_min = 0, z_max = 10000, zlab = "Altitude ft", ...
title = "3D Trajectory", line_color = "blue"}})?
plots.att_graph3D=plot(p(1,1), p(2,1), -p(3,1), {line_color = "green", keep})?
plots.att_graph3D=plot(q(1,1), q(2,1), -q(3,1), {line_color = "red", keep, ...
legend = ["Vehicle Trajectory", "No Control Trajectory", "CARP Trajectory"], ...
xlab="X", ylab="Y"}})?

# make plots of interesting data
execute file = "controls";

delete predicted_x predicted_y predicted_z;
delete TOP_mat controlled_x controlled_y
delete final_con_x final_con_y

```

2. Math script file "controls.ms"

```
# { This batch file plots the affect of the control system for TOP }#

# state of PMA's plot
plots.contPMAplot=plot(y(13,1), {rows=4, columns=1, row=1, column=1, ylab="PMA1_cmd psi", ...
    xlab="Time sec", xmax=360})?
plots.contPMAplot=plot(y(14,1), {row=2, column=1, ylab="PMA2_cmd psi", xlab="Time sec", ...
    xmax=360})?
plots.contPMAplot=plot(y(15,1), {row=3, column=1, ylab="PMA3_cmd psi", xlab="Time sec", ...
    xmax=360})?
plots.contPMAplot=plot(y(16,1), {row=4, column=1, ylab="PMA4_cmd psi", xlab="Time sec", ...
    xmax=360})?

#plot x_error norm and y_error norm (must be >abs(0.3) for a control)
plots.controlplot=plot(y(28,1), {rows=4, columns=1, row=1, column=1, ...
    ylab="Norm of X Error", xlab="Time sec", xmax=360})?
plots.controlplot=plot(t, 0.3*ones(length(t),1), {row=1, column=1, keep})?
plots.controlplot=plot(t, -0.3*ones(length(t),1), {row=1, column=1, keep})?
plots.controlplot=plot(y(29,1), {row=2, column=1, ylab="Norm of Y Error", ...
    xlab="Time sec", xmax=360})?
plots.controlplot=plot(t, 0.3*ones(length(t),1), {row=2, column=1, keep})?
plots.controlplot=plot(t, -0.3*ones(length(t),1), {row=2, column=1, keep})?

#plot tolerance cone with radial error
plots.controlplot=plot(y(30,1), {row=3, column=1, ylab="Tolerance with Radial Error ft", ...
    xlab="Time sec", xmax=360})?
plots.controlplot=plot(y(31,1), {row=3, column=1, line_color="red", ...
    legend=["Tolerance", "Radial Error"], keep, xmax=360})?

#now plot PMA positions
plots.controlplot=plot(y(17,1), {row=4, column=1, xlab="Time sec", line_color="red", ...
    keep, xmax=360})?
plots.controlplot=plot(y(18,1), {row=4, column=1, xlab="Time sec", line_color="green", ...
    keep, xmax=360})?
plots.controlplot=plot(y(19,1), {row=4, column=1, xlab="Time sec", line_color="blue", ...
    keep, xmax=360})?
plots.controlplot=plot(y(20,1), {row=4, column=1, ylab="PMA pos psi", xlab="Time sec", ...
    line_color="black", legend=["PMA1", "PMA2", "PMA3", "PMA4"], keep, xmax=360})?

#plot no. of controls with glide ratio
plots.controlgrplot=plot(y(21,1))?
plots.controlgrplot=plot(y(25,1), {keep, ylab="No. of Controls and Glide Ratio", ...
    xlab="Time sec", xmax=360, ymax=3, legend=["Glide Ratio", "No. of Controls"]})?

#plot actuator data
#first three plots are comparison of fill cycles with "counter" and
#fill cycles with the data extrapolator and threshold counter
#initial four actuations on parachute release are included
#next two plots are fill time and reservoir pressure respectively
plots.actuplot=plot(y(26,1)+4, {rows=5, columns=1, row=1, column=1, ...
    ylab="Fill Cycles using Counter", xlab="Time sec", xmax=360})?
plots.actuplot=plot(y(33,1), {row=2, column=1, ylab="Fill Cycles using Data", ...
```



```

    xlab="Time sec", xmax=360}}?
plots.actuplot=plot(y(42,1)+4, {row=3, column=1, ylab="Fill Cycles using Threshold", ...
    xlab="Time sec", xmax=360}}?
plots.actuplot=plot(y(27,1), {row=4, column=1, ylab="Fill Time sec", xlab="Time sec", ...
    xmax=360}}?
plots.actuplot=plot(y(32,1), {row=5, column=1, ylab="Reservoir Pressure psi", ...
    xlab="Time sec", xmax=360}}?

#plot results of GPS and sensor information
plots.GPSplot=plot(y(1,1), {rows=3, columns=1, row=1, column=1, ...
    ylab="True X Position and GPS X ft", xlab="Time sec", xmax=360}}?
plots.GPSplot=plot(y(22,1), {row=1, column=1, keep, line_color="blue", ...
    legend=["True Position", "Sensor Position"], xmax=360}}?
plots.GPSplot=plot(y(2,1), {row=2, column=1, ylab="True Y Position and GPS Y ft", ...
    xlab="Time sec", xmax=360}}?
plots.GPSplot=plot(y(23,1), {row=2, column=1, keep, line_color="red", ...
    legend=["True Position", "Sensor Position"], xmax=360}}?
plots.GPSplot=plot(-y(3,1), {row=3, column=1, ylab="True Z Position and GPS Z ft", ...
    xlab="Time sec", xmax=360}}?
plots.GPSplot=plot(-y(24,1), {row=3, column=1, keep, line_color="green", ...
    legend=["True Position", "Sensor Position"], xmax=360}}?

plots.GPSerrplot=plot(y(34,1)+y(37,1), {rows=3, columns=1, row=1, column=1, ...
    ylab="GPS X Error ft", xlab="Time sec", xmax=360}}?
plots.GPSerrplot=plot(y(35,1)+y(38,1), {row=2, column=1, ylab="GPS Y error ft", ...
    xlab="Time sec", xmax=360}}?
plots.GPSerrplot=plot(y(36,1)+y(39,1), {row=3, column=1, ylab="GPS Z error ft", ...
    xlab="Time sec", xmax=360}}?

#set up compass so it is between 000 and 360
[rtrue,mtrue]=mod(y(6,1),2*pi);
[rsens,msens]=mod(y(40,1),2*pi);

plots.compassplot=plot(180/pi*rtrue, {rows=2, columns=1, row=1, column=1, ...
    ylab="True Heading and Compass Heading deg", xlab="Time sec", xmax=360}}?
plots.compassplot=plot(180/pi*rsens, {row=1, column=1, keep, line_color="blue", ...
    legend=["True Heading", "Sensor Heading"], xmax=360}}?
plots.compassplot=plot(180/pi*y(41,1), {row=2, column=1, ylab="Heading Sensor Error deg", ...
    xlab="Time sec", xmax=360}}?

delete rtrue mtrue rsens msens

delete q y p
delete control_error
delete saseed1 saseed2 saseed3 saseed4 saseed5 saseed6 saseed7 saseed8 saseed9 saseed

```

3. Math scrip file "agesims.ms"

```
{ batchfile: This batchfile runs the CARP predictor at hour0. Then it runs
  TOP and NOFORECAST for each hour wind file after that and plots them on one plot
  to do comparisons
}#

# colors for plots
colors=["black", "blue", "green", "orange", "aquamarine", "copper", "melon", "forest", ...
  "brick", "gold"];

# set actuator data for both CARP and TOP (or NOFORECAST)
pma_max_pressure = 175;
pma_max_pressures = [175,175,175,175];
del_p_vs_p=actuators.del_p_vs_p175;
fill_time_vs_p=actuators.fill_time_vs_p175;
del_res_per_PMA=actuators.del_res_per_PMA175;
fills_vs_respress=actuators.fills_vs_respress175;
deflate_time=actuators.deflate_time175;
init_reservoir_press=actuators.init_reservoir_press175;

# set CARP wind files--will always be this first hour
wind.forecast_alt=wind.Wind0(:,1);
wind.forecast_x=wind.Wind0(:,3);
wind.forecast_y=wind.Wind0(:,2);
wind.forecast_z=wind.Wind0(:,4);

# sim CARP
q=sim("CARP", t, {ialg="VKM"});

# create predicted trajectory for TOP
pred_mat=makematrix(q, {channels});
predicted_x=pred_mat(:,1);
predicted_y=pred_mat(:,2);
predicted_z=pred_mat(:,3);

# create target position for NOFORECAST
target_x=[predicted_x(length(predicted_x)),predicted_x(length(predicted_x))];
target_y=[predicted_y(length(predicted_y)),predicted_y(length(predicted_y))];

final_CARP_x=predicted_x(length(predicted_x));
final_CARP_y=predicted_y(length(predicted_y));

delete pred_mat;
# done with the CARP simulation

# initial plots of CARP, and target position moved to (0,0) on the ground
plots.TOPgraph2D=plot(q(1,1)-final_CARP_x, q(2,1)-final_CARP_y, {x_min = -7500, ...
  x_max = 7500, y_min = -7500, y_max = 7500, title = "FORECAST2D", line_color = "red", ...
  xlab="X", ylab="Y"});

plots.TOPgraph3D=plot(q(1,1)-final_CARP_x, q(2,1)-final_CARP_y, -q(3,1), {x_min = -7500, ...
  x_max = 7500, y_min = -7500, y_max = 7500, z_min = 0, z_max = 10000, ...
```

```

zlab = "Altitude ft", title = "FORECAST3D", line_color = "red", xlab="X", ylab="Y"))?

plots.NOFOREgraph2D=plot(0, 0, {x_min = -7500, x_max = 7500, y_min = ...
-7500, y_max = 7500, title = "NOFORECAST2D", xlab="X", ylab="Y", line=0, marker=1, ...
marker_style="x", marker_color="red"})?

plots.NOFOREgraph3D=plot(zeros(length(q(3,1)),1), zeros(length(q(3,1)),1), ...
-q(3,1), {x_min = -7500, x_max = 7500, y_min = -7500, y_max = 7500, z_min = 0, ...
z_max = 10000, zlab = "Altitude ft", title = "NOFORECAST3D", line_color = "red", ...
xlab="X", ylab="Y"})?

# now to the actual TOP simulation

for i=1:10

# set up winds for actual simulation
wind.newwind=wind.windlist(i+1);
wind.actual_alt=wind.newwind(:,1)';
wind.actual_x=wind.newwind(:,3)';
wind.actual_y=wind.newwind(:,2)';
wind.actual_z=wind.newwind(:,4)';

# set GPS random number seeds
set distribution uniform;
saseed=round(9999*random(1,9),{up});
saseed1=saseed(1); saseed2=saseed(2); saseed3=saseed(3); saseed4=saseed(4);
saseed5=saseed(5); saseed6=saseed(6); saseed7=saseed(7); saseed8=saseed(8);
saseed9=saseed(9);

# selective availability on or off (0 is off, 1 is on)
saon=0; # equal to 0 always now because of President's declaration

# random compass bias between -2 and 2 degrees
compass_bias=4*(random(1,1)-0.5);

# set offset of vehicle trajectory (zero for these simulations)
x_offset=0;
y_offset=0;
linear_position_init = [x_offset; y_offset; drop_alt];

# sim TOP model
y=sim("TOP", t, {ialg="VKM"});

# calculations for TOP
TOP_mat=makematrix(y,{channels});
controlled_x=TOP_mat(:,1);
controlled_y=TOP_mat(:,2);
final_con_x=controlled_x(length(controlled_x));
final_con_y=controlled_y(length(controlled_y));

simresults7.control_error(i)=((final_con_x-final_CARP_x)**2+(final_con_y-final_CARP_y)**2)**0.5;

# sim the NOFORECAST model

```

```

p=sim("NOFORECAST", t, {ialg="VKM"});

# calculations for NOFORECAST
NOFORE_mat=makematrix(p,{channels});
nofore_x=NOFORE_mat(:,1);
nofore_y=NOFORE_mat(:,2);
final_nofore_x=nofore_x(length(nofore_x));
final_nofore_y=nofore_y(length(nofore_y));

simresults7.nofore_error(i)=((final_nofore_x-final_CARP_x)**2+(final_nofore_y-
final_CARP_y)**2)**0.5;

# plots
plots.TOPgraph2D=plot(y(1,1)-final_CARP_x, y(2,1)-final_CARP_y, {keep=plots.TOPgraph2D, ...
line_color = colors(i)});

plots.TOPgraph3D=plot(y(1,1)-final_CARP_x, y(2,1)-final_CARP_y, -y(3,1), ...
{line_color = colors(i), keep=plots.TOPgraph3D});

plots.NOFOREgraph2D=plot(p(1,1)-final_CARP_x, p(2,1)-final_CARP_y, ...
{keep=plots.NOFOREgraph2D, line_color = colors(i)});

plots.NOFOREgraph3D=plot(p(1,1)-final_CARP_x, p(2,1)-final_CARP_y, -p(3,1), ...
{line_color = colors(i), keep=plots.NOFOREgraph3D});

# end the simulation
endfor;

# put the legends in plots
plots.TOPgraph2D=plot({legend=["CARP Trajectory", "Hour1", "Hour2", "Hour3", "Hour4", ...
"Hour5", "Hour6", "Hour7", "Hour8", "Hour9", "Hour10"],keep=plots.TOPgraph2D});

plots.TOPgraph3D=plot({legend=["CARP Trajectory", "Hour1", "Hour2", "Hour3", "Hour4", ...
"Hour5", "Hour6", "Hour7", "Hour8", "Hour9", "Hour10"],keep=plots.TOPgraph3D});

plots.NOFOREgraph2D=plot({legend=["CARP Trajectory", "Hour1", "Hour2", "Hour3", "Hour4", ...
"Hour5", "Hour6", "Hour7", "Hour8", "Hour9", "Hour10"],keep=plots.NOFOREgraph2D});

plots.NOFOREgraph3D=plot({legend=["CARP Trajectory", "Hour1", "Hour2", "Hour3", "Hour4", ...
"Hour5", "Hour6", "Hour7", "Hour8", "Hour9", "Hour10"],keep=plots.NOFOREgraph3D});

delete predicted_x predicted_y predicted_z;
delete TOP_mat controlled_x controlled_y
delete final_con_x final_con_y
delete final_CARP_x final_CARP_y
delete q y
delete colors target_x target_y i
delete p NOFORE_mat nofore_x nofore_y final_nofore_x final_nofore_y saseed
delete saseed1 saseed2 saseed3 saseed4 saseed5 saseed6 saseed7 saseed8 saseed9 saseed

```

4. Math script file "runsim.ms"

```
{ batchfile: runsim.ms UPDATED 20 Apr 00 1430
  This batch file first runs the CARP predictor.
  This batch file is also designed to create the predicted_x, predicted_y,
  and predicted_z row matrices that are plugged into the predicted x
  and predicted y linear interpolation blocks in Controller. The file then
  runs TOP, NOFORECAST, and NOCONTROL and calculates data}#

# sim CARP
q=sim("CARP", t, {ialg="VKM"});

# make predicted trajectory for TOP
pred_mat=makematrix(q,{channels});
predicted_x=pred_mat(:,1);
predicted_y=pred_mat(:,2);
predicted_z=pred_mat(:,3);

delete pred_mat;

# target position for NOFORECAST
target_x=[predicted_x(length(predicted_x)),predicted_x(length(predicted_x))];
target_y=[predicted_y(length(predicted_y)),predicted_y(length(predicted_y))];

# sim the three models
y=sim("TOP", t, {ialg="VKM"});

p=sim("NOFORECAST", t, {ialg="VKM"});

r=sim("NOCONTROL", t, {ialg="VKM"});

execute file="fix_out" # should run this to fix the data before plotting
execute file="plot_traj2D" # run 2-D plot, not run in big simulations
execute file="plot_traj3D" # run 3-D plot, not run in big simulations

TOP_mat=makematrix(y,{channels});
NOCONTROL_mat=makematrix(r,{channels});

controlled_x=TOP_mat(:,1);
controlled_y=TOP_mat(:,2);

no_controlled_x=NOCONTROL_mat(:,1);
no_controlled_y=NOCONTROL_mat(:,2);

final_con_x=controlled_x(length(controlled_x));
final_con_y=controlled_y(length(controlled_y));

final_nocon_x=no_controlled_x(length(no_controlled_x));
final_nocon_y=no_controlled_y(length(no_controlled_y));

final_CARP_x=CARP_x(length(CARP_x));
```

```

final_CARP_y=CARP_y(length(CARP_y));

con_act=TOP_mat(length(TOP_mat(:,26)),26);

NOFORE_mat=makematrix(p,{channels});
nofore_x=NOFORE_mat(:,1);
nofore_y=NOFORE_mat(:,2);
final_nofore_x=nofore_x(length(nofore_x));
final_nofore_y=nofore_y(length(nofore_y));
nofore_act=NOFORE_mat(length(NOFORE_mat(:,26)),26);

no_control_error=((final_nocon_x-final_CARP_x)**2+(final_nocon_y-final_CARP_y)**2)**0.5;
control_error=((final_con_x-final_CARP_x)**2+(final_con_y-final_CARP_y)**2)**0.5;
nofore_error=((final_nofore_x-target_x(1,1))**2+(final_nofore_y-target_y(1,1))**2)**0.5;

```

5. Math script file "fix_out.ms"

```

#{ batchfile: fix_out.ms UPDATED 20 Apr 00 1430
  This batch file is designed to fix the output of the 3-dof
  rigid-body parachute model. This batch file assumes that
  simulation results have been written to pdm y with x, y, and
  z-components of position being the 1st, 2nd, and 3rd rows of
  each sub-matrix of the pdm. This batch file also assumes that the
  output of the CARP trajectory predictor has been written to workspace variables
  predicted_x, predicted_y, predicted_z. Another assumption is that the output
  of the model with controller has been written to PDM y, the output of NOFORECAST
  has been written to PDM p, and output of NOCONTROL has been written to PDM r}#

IF y(3,1,1) < 0 THEN
  y(3,1) = -y(3,1); # change sign of altitude variable for plotting actual trajectory
ENDIF

IF p(3,1,1) < 0 THEN # change sign of altitude variable for plotting NOFORECAST trajectory
  p(3,1) = -p(3,1);
ENDIF

IF r(3,1,1) < 0 THEN # change sign of altitude variable for plotting NOCONTROL trajectory
  r(3,1) = -r(3,1);
ENDIF

IF predicted_z(1,1) < 0 THEN
  CARP_z = -predicted_z'; # change sign of altitude variable for plotting
                          # CARP trajectory
                          # need to use a new name for this variable so not
                          # to screw up values of predicted_x, etc.
ELSE
  CARP_z = predicted_z';
ENDIF
CARP_x = predicted_x';
CARP_y = predicted_y';

```

6. Math script file "runmanysimsnewwind.ms"

This batch file runs the control strategy comparison simulations with the 11 different
wind files collected hourly from YPG

set seed 128; # random number seed (use same number for same results)

set actuator data
pma_max_pressure = 175;
pma_max_pressures = [175,175,175,175];
del_p_vs_p=actuators.del_p_vs_p175;
fill_time_vs_p=actuators.fill_time_vs_p175;
del_res_per_PMA=actuators.del_res_per_PMA175;
fills_vs_respress=actuators.fills_vs_respress175;
deflate_time=actuators.deflate_time175;
init_reservoir_press=actuators.init_reservoir_press175;

start the iterations

for i=1:1000

must change initial position for TOP model, for NO CONTROL model, and for
NOFORECAST model (they should be the same initial position called "linear_
position_init") for each iteration.

set max offset
max_offset=2500/sqrt(2);

set x offset and y offset
set distribution normal; # normal distribution for offsets

x_offset = max_offset * (random(1,1))/4;
IF x_offset == 0 THEN
x_offset = 0.1;
ENDIF

y_offset = max_offset * (random(1,1))/4;
IF y_offset == 0 THEN
y_offset = 0.1;
ENDIF

linear_position_init = [x_offset; y_offset; drop_alt];

must change the wind files for actual wind for each iteration
the forecasted wind is chosen from wind files earlier than the actual wind
set distribution uniform; # a uniform distribution for forecast wind choices

IF i <= 100 THEN
wind.newwind = wind.windlist(2);
wind.actual_alt = wind.newwind(:,1)';
wind.actual_x = wind.newwind(:,3)';
wind.actual_y = wind.newwind(:,2)';

```

wind.actual_z = wind.newwind(:,4);
forecast_wind_choice = round(1*random(1,1),{up}); # choose a random number 1-1
wind.forecastwind = wind.windlist(forecast_wind_choice);
wind.forecast_alt = wind.forecastwind(:,1);
wind.forecast_x = wind.forecastwind(:,3);
wind.forecast_y = wind.forecastwind(:,2);
wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 200 THEN
    wind.newwind = wind.windlist(3);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(2*random(1,1),{up}); # choose a random number 1-2
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 300 THEN
    wind.newwind = wind.windlist(4);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(3*random(1,1),{up}); # choose a random number 1-3
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 400 THEN
    wind.newwind = wind.windlist(5);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(4*random(1,1),{up}); # choose a random number 1-4
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 500 THEN
    wind.newwind = wind.windlist(6);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(5*random(1,1),{up}); # choose a random number 1-5
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);

```



```

wind.forecast_y = wind.forecastwind(:,2);
wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 600 THEN
    wind.newwind = wind.windlist(7);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(6*random(1,1),{up}); # choose a random number 1-6
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 700 THEN
    wind.newwind = wind.windlist(8);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(7*random(1,1),{up}); # choose a random number 1-7
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 800 THEN
    wind.newwind = wind.windlist(9);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(8*random(1,1),{up}); # choose a random number 1-8
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 900 THEN
    wind.newwind = wind.windlist(10);
    wind.actual_alt = wind.newwind(:,1);
    wind.actual_x = wind.newwind(:,3);
    wind.actual_y = wind.newwind(:,2);
    wind.actual_z = wind.newwind(:,4);
    forecast_wind_choice = round(9*random(1,1),{up}); # choose a random number 1-9
    wind.forecastwind = wind.windlist(forecast_wind_choice);
    wind.forecast_alt = wind.forecastwind(:,1);
    wind.forecast_x = wind.forecastwind(:,3);
    wind.forecast_y = wind.forecastwind(:,2);
    wind.forecast_z = wind.forecastwind(:,4);
ELSEIF i <= 1000 THEN
    wind.newwind = wind.windlist(11);
    wind.actual_alt = wind.newwind(:,1);

```

```

wind.actual_x = wind.newwind(:,3)';
wind.actual_y = wind.newwind(:,2)';
wind.actual_z = wind.newwind(:,4)';
forecast_wind_choice = round(10*random(1,1),{up}); # choose a random number 1-10
wind.forecastwind = wind.windlist(forecast_wind_choice);
wind.forecast_alt = wind.forecastwind(:,1)';
wind.forecast_x = wind.forecastwind(:,3)';
wind.forecast_y = wind.forecastwind(:,2)';
wind.forecast_z = wind.forecastwind(:,4)';
ENDIF
# set GPS random number seeds
set distribution uniform; # uniform distribution for GPS seeds and compass biases
saseed=round(9999*random(1,9),{up});
saseed1=saseed(1); saseed2=saseed(2); saseed3=saseed(3); saseed4=saseed(4);
saseed5=saseed(5); saseed6=saseed(6); saseed7=saseed(7); saseed8=saseed(8);
saseed9=saseed(9);

# random compass bias between -2 and 2 degrees
compass_bias=4*(random(1,1)-0.5);

# now you can run a simulation

execute file = "runsim";

# calculate more needed data for polar plots
control_error_x = final_con_x - final_CARP_x;
control_error_y = final_con_y - final_CARP_y;
nofore_error_x = final_nofore_x - final_CARP_x;
nofore_error_y = final_nofore_y - final_CARP_y;
control_error_angle = atan2(control_error_y, control_error_x)*(180/pi);
nofore_error_angle = atan2(nofore_error_y, nofore_error_x)*(180/pi);
releasept_x = x_offset - final_CARP_x;
releasept_y = y_offset - final_CARP_y;
releasept_angle = atan2(releasept_y, releasept_x)*(180/pi);
releasept_radius = (releasept_y**2+releasept_x**2)**0.5;
CARPrelease_x = 0 - final_CARP_x;
CARPrelease_y = 0 - final_CARP_y;
CARPrelease_angle = atan2(CARPrelease_y, CARPrelease_x)*(180/pi);
CARPrelease_radius = (CARPrelease_y**2+CARPrelease_x**2)**0.5;

# form huge ix22 data matrix

simresults8.data(i,:) = [x_offset, y_offset, forecast_wind_choice, no_control_error, ...
control_error, nofore_error, con_act, nofore_act, control_error_y, control_error_x, ...
control_error_angle, nofore_error_y, nofore_error_x, nofore_error_angle, releasept_y, ...
releasept_x, releasept_angle, releasept_radius, CARPrelease_y, CARPrelease_x, ...
CARPrelease_angle, CARPrelease_radius];

# WHAT ITERATION ARE WE ON???
display("Run No. = "); display(i)

delete CARP_x CARP_y CARP_z predicted_x predicted_y predicted_z;
delete TOP_mat controlled_x controlled_y no_controlled_x no_controlled_y

```

```

delete final_con_x final_con_y final_nocon_x final_nocon_y final_nofore_x final_nofore_y
delete final_CARP_x final_CARP_y
delete p q y r target_x target_y nofore_x nofore_y NOFORE_mat
delete no_control_error control_error nofore_error con_act nofore_act
delete NOCONTROL_mat forecast_wind_choice x_offset y_offset i
delete saseed saseed1 saseed2 saseed3 saseed4 saseed5 saseed6 saseed7 saseed8 saseed9
delete control_error_x control_error_y nofore_error_x nofore_error_y control_error_angle
delete releasept_x releasept_y releasept_angle releasept_radius CARPrelease_x
delete CARPrelease_y nofore_error_angle CARPrelease_angle CARPrelease_radius

# repeat the process for the specified number of iterations

endfor;

```

7. Math script file "runmanysimsandplot0.ms"

```

# This batch file runs the simulation where the predicted wind chosen was the wind file
# from the 0800 balloon launch. The actual winds are chosen at random from later wind
# files. Both TOP and NOFORECAST are run and plotted for each iteration.

set seed 128; # random number seed (use same number for same results)

# set actuator data
pma_max_pressure = 175;
pma_max_pressures = [175,175,175,175];
del_p_vs_p=actuators.del_p_vs_p175;
fill_time_vs_p=actuators.fill_time_vs_p175;
del_res_per_PMA=actuators.del_res_per_PMA175;
fills_vs_respress=actuators.fills_vs_respress175;
deflate_time=actuators.deflate_time175;
init_reservoir_press=actuators.init_reservoir_press175;

# choose a forecasted wind to follow
j=3; # choose from wind files Wind0 to Wind9 (Wind0 is j=1, Wind9 is j=10)
wind.forecastwind=wind.windlist(j);
wind.forecast_alt = wind.forecastwind(:,1)';
wind.forecast_x = wind.forecastwind(:,3)';
wind.forecast_y = wind.forecastwind(:,2)';
wind.forecast_z = wind.forecastwind(:,4)';

# set up the predicted trajectory

q=sim("CARP", t, {ialg="VKM"});

# create predicted trajectory for TOP
pred_mat=makematrix(q,{channels});
predicted_x=pred_mat(:,1)';
predicted_y=pred_mat(:,2)';
predicted_z=pred_mat(:,3)';

```

```

delete pred_mat;

final_CARP_x=predicted_x(length(predicted_x));
final_CARP_y=predicted_y(length(predicted_y));

# create target position for NOFORECAST
target_x=[predicted_x(length(predicted_x)),predicted_x(length(predicted_x))];
target_y=[predicted_y(length(predicted_y)),predicted_y(length(predicted_y))];

# initial plots of CARP, target position at (0,0) on the ground
q(1,1) = q(1,1) - final_CARP_x;
q(2,1) = q(2,1) - final_CARP_y;

plots.TOPgraph2D=plot(q(1,1), q(2,1), {x_min = -10000, x_max = 10000, ...
y_min = -10000, y_max = 10000, title = "TRAJECTORYSEEK2D", line_color = "red", ...
xlab="South-North", ylab="West-East", line_width=4});

plots.TOPgraph3D=plot(q(1,1), q(2,1), ...
-q(3,1), {x_min = 10000, x_max = -10000, y_min = ...
-10000, y_max = 10000, z_min = 0, z_max = 10000, zlab = "Altitude ft", ...
title = "TRAJECTORYSEEK3D", line_color = "red", line_width=4, xlab="South-North", ...
ylab="West-East"});

plots.NOFOREgraph2D=plot(0, 0, {x_min = -10000, x_max = 10000, y_min = ...
-10000, y_max = 10000, title = "TARGETSEEK2D", line=0, marker=1, marker_style = "x", ...
marker_color = "red", xlab="South-North", ylab="West-East"});

plots.NOFOREgraph3D=plot(zeros(length(q(3,1)),1), zeros(length(q(3,1)),1), ...
-q(3,1), {x_min = 10000, x_max = -10000, y_min = -10000, y_max = 10000, z_min = 0, ...
z_max = 10000, zlab = "Altitude ft", title = "TARGETSEEK3D", line_color = "red", ...
line_width=4, xlab="South-North", ylab="West-East"});

# start the iterations

for i=1:30

# must change initial position for TOP model, for NO CONTROL model, and for
# NOFORECAST model (they should be the same initial position called "linear_
# position_init") for each iteration.

# set max offset

max_offset=2500/sqrt(2);

# set x offset and y offset
set distribution normal; # normal distribution for offsets

x_offset = max_offset * random(1,1)/4;
IF x_offset == 0 THEN
    x_offset = 0.1;
ENDIF

y_offset = max_offset * random(1,1)/4;

```

```

IF y_offset == 0 THEN
    y_offset = 0.1;
ENDIF

linear_position_init = [x_offset; y_offset; drop_alt];

# must change the wind files for actual wind for each iteration

set distribution uniform; # a uniform distribution for actual wind choices

actual_wind_choice = round((11-j)*random(1,1)+j,{up}); # choose an actual wind file of a later
wind.actualwind = wind.windlist(actual_wind_choice); # hour than the forecasted file
wind.actual_alt = wind.actualwind(:,1);
wind.actual_x = wind.actualwind(:,3);
wind.actual_y = wind.actualwind(:,2);
wind.actual_z = wind.actualwind(:,4);

# set GPS random number seeds
set distribution uniform; # uniform distribution for GPS seeds and compass biases
saseed=round(9999*random(1,9),{up});
saseed1=saseed(1); saseed2=saseed(2); saseed3=saseed(3); saseed4=saseed(4);
saseed5=saseed(5); saseed6=saseed(6); saseed7=saseed(7); saseed8=saseed(8);
saseed9=saseed(9);

# random compass bias between -2 and 2 degrees
set distribution uniform;
compass_bias=4*(random(1,1)-0.5);

# now you can run a simulation
y=sim("TOP", t, {ialg="VKM"});

p=sim("NOFORECAST", t, {ialg="VKM"});

#change frame of reference so that desired target position is at the origin
y(1,1) = y(1,1) - final_CARP_x;
y(2,1) = y(2,1) - final_CARP_y;
p(1,1) = p(1,1) - final_CARP_x;
p(2,1) = p(2,1) - final_CARP_y;

plots.TOPgraph2D=plot(y(1,1), y(2,1), {keep=plots.TOPgraph2D, line_color="blue", ...
    line_width=4});

plots.TOPgraph3D=plot(y(1,1), y(2,1), -y(3,1), {keep=plots.TOPgraph3D, line_color="blue", ...
    line_width=4});

plots.NOFOREgraph2D=plot(p(1,1), p(2,1), {keep=plots.NOFOREgraph2D, line_color="black", ...
    line_width=4});

plots.NOFOREgraph3D=plot(p(1,1), p(2,1), -p(3,1), {keep=plots.NOFOREgraph3D, ...
    line_color="black", line_width=4});

```

```

# repeat the process for the specified number of iterations

# WHAT ITERATION ARE WE ON???
display("Run No. = "); display(i)

endfor;

# circle
theta = [0:0.1:2*pi];
x_circ = 328.08*cos(theta);
y_circ = 328.08*sin(theta);
plot(x_circ,y_circ,zeros(1,63),{keep=plots.TOPgraph3D, line=1, line_color="green", ...
    line_width=3})?
plot(x_circ,y_circ,zeros(1,63),{keep=plots.NOFOREgraph3D, line=1, line_color="green", ...
    line_width=3})?
plot(x_circ,y_circ,{keep=plots.TOPgraph2D, line=1, line_color="green", line_width=3})?
plot(x_circ,y_circ,{keep=plots.NOFOREgraph2D, line=1, line_color="green", line_width=3})?

# delete unneeded variables
delete x_circ y_circ theta
delete predicted_x predicted_y predicted_z;
delete final_CARP_x final_CARP_y
delete p q y target_x target_y
delete i j actual_wind_choice wind.forecastwind wind.actualwind
delete saseed1 saseed2 saseed3 saseed4 saseed5 saseed6 saseed7 saseed8 saseed9

```

8. Math script file "runactsim2.ms"

```

#{ batchfile:
    This batch file first assumes the CARP predictor with your wind file has been run.
    It also assumes that the "fix-out" file has been run to transform the data from the CARP
    model into predicted_x, _y, and _z matrices to be plugged into the Controller.
    The batch file then runs the TOP sim. Relevant information is then calculated. For actuator
    study }#

y=sim("TOP", t, {ialg="VKM"});

TOP_mat=makematrix(y,{channels});

controlled_x=TOP_mat(:,1);
controlled_y=TOP_mat(:,2);

final_con_x=controlled_x(length(controlled_x));
final_con_y=controlled_y(length(controlled_y));

final_CARP_x=CARP_x(length(CARP_x));
final_CARP_y=CARP_y(length(CARP_y));

con_act=TOP_mat(length(TOP_mat(:,26)),26);
max_pma_fill_time=TOP_mat(length(TOP_mat(:,27)),27);
control_error=((final_con_x-final_CARP_x)**2+(final_con_y-final_CARP_y)**2)**0.5;

```

```
delete TOP_mat controlled_x controlled_y final_con_x final_con_y y;
```

9. Math script file "runmanyactsims2.ms"

```
# This batch file runs the actuator study with the "worst", "worse", "average", "better"  
# and "best" actuator models. 100 iterations are run with each and data is collected.  
# Every simulation used the TOP model
```

```
set seed 32;                                # random number seed (use same number for same results)
```

```
# must NOT change the wind files for actual wind for each iteration
```

```
# for this file the winds were just set
```

```
actual_wind_choice = 8;  
wind.newwind = wind.windlist(actual_wind_choice);  
wind.actual_alt = wind.newwind(:,1);  
wind.actual_x = wind.newwind(:,3);  
wind.actual_y = wind.newwind(:,2);  
wind.actual_z = wind.newwind(:,4);
```

```
# choose the forecasted winds for ALL iterations
```

```
# for this file the winds were just set
```

```
forecast_wind_choice = 6;                    # DO NOT choose a random number 1-1  
wind.forecastwind = wind.windlist(forecast_wind_choice);  
wind.forecast_alt = wind.forecastwind(:,1);  
wind.forecast_x = wind.forecastwind(:,3);  
wind.forecast_y = wind.forecastwind(:,2);  
wind.forecast_z = wind.forecastwind(:,4);
```

```
# run CARP one time first
```

```
q=sim("CARP", t, {ialg="VKM"});
```

```
# create the predicted trajectory
```

```
pred_mat=makematrix(q,{channels});  
predicted_x=pred_mat(:,1);  
predicted_y=pred_mat(:,2);  
predicted_z=pred_mat(:,3);
```

```
delete pred_mat q;
```

```
CARP_x = predicted_x';  
CARP_y = predicted_y';
```

```
# start the iterations
```

```
for i=1:100
```

```

# must change initial position for TOP model,
# initial position called "linear_
# position_init") for each iteration

set distribution normal; # normal distribution for release point offset

max_offset=2500/sqrt(2);

x_offset = max_offset * (random(1,1))/4;
IF x_offset == 0 THEN
  x_offset = 0.1;
ENDIF

y_offset = max_offset * (random(1,1))/4;
IF y_offset == 0 THEN
  y_offset = 0.1;
ENDIF

linear_position_init = [x_offset; y_offset; drop_alt];

# set GPS random number seeds
set distribution uniform; # uniform distribution for GPS seeds and compass biases
saseed=round(9999*random(1,9),{up});
saseed1=saseed(1); saseed2=saseed(2); saseed3=saseed(3); saseed4=saseed(4);
saseed5=saseed(5); saseed6=saseed(6); saseed7=saseed(7); saseed8=saseed(8);
saseed9=saseed(9);

# random compass bias between -2 and 2 degrees
compass_bias=4*(random(1,1)-0.5);

# now, test different actuator sets
# this will not change from iteration to iteration

pma_max_pressure = 175;
pma_max_pressures = [175,175,175,175];
del_p_vs_p = actuators.del_p_vs_p175;
fills_vs_respress=actuators.fills_vs_respress175;
deflate_time=actuators.deflate_time175;
init_reservoir_press=actuators.init_reservoir_press175;
del_res_per_PMA = actuators.del_res_per_PMA175;

# this will be changed for the different controllers
fill_time_vs_p = actuators.worst;

# now you can run a simulation
# we will use runactsim2
execute file = "runactsim2";

# this is the data we collect on each run
control_error_worst = control_error;
con_act_worst = con_act;
max_pma_fill_time_worst = max_pma_fill_time;

```



```
# do it for each of the four remaining actuator models
```

```
fill_time_vs_p = actuators.worse;  
execute file = "runactsim2";  
control_error_worse = control_error;  
con_act_worse = con_act;  
max_pma_fill_time_worse = max_pma_fill_time;
```

```
fill_time_vs_p = actuators.average;  
execute file = "runactsim2";  
control_error_average = control_error;  
con_act_average = con_act;  
max_pma_fill_time_average = max_pma_fill_time;
```

```
fill_time_vs_p = actuators.better;  
execute file = "runactsim2";  
control_error_better = control_error;  
con_act_better = con_act;  
max_pma_fill_time_better = max_pma_fill_time;
```

```
fill_time_vs_p = actuators.best;  
execute file = "runactsim2";  
control_error_best = control_error;  
con_act_best = con_act;  
max_pma_fill_time_best = max_pma_fill_time;
```

```
##### five runs of TOP for each iteration#####
```

```
# this is the data output
```

```
simresults9.data(i,:) = [x_offset, y_offset, control_error_worst, con_act_worst, ...  
    max_pma_fill_time_worst, control_error_worse, con_act_worse, max_pma_fill_time_worse, ...  
    control_error_average, con_act_average, max_pma_fill_time_average, ...  
    control_error_better, con_act_better, max_pma_fill_time_better, ...  
    control_error_best, con_act_best, max_pma_fill_time_best, actual_wind_choice, ...  
    forecast_wind_choice];
```

```
delete x_offset y_offset control_error_worst con_act_worst max_pma_fill_time_worst;  
delete control_error_worse con_act_worse max_pma_fill_time_worse;  
delete control_error_average con_act_average max_pma_fill_time_average;  
delete control_error_better con_act_better max_pma_fill_time_better;  
delete control_error_best con_act_best max_pma_fill_time_best;  
delete saseed saseed1 saseed2 saseed3 saseed4 saseed5 saseed6 saseed7 saseed8 saseed9  
delete control_error con_act max_pma_fill_time
```

```
# repeat the process for the specified number of iterations
```

```
# WHAT ITERATION ARE WE ON???
```

```
display("Run No. = "); display(i)
```

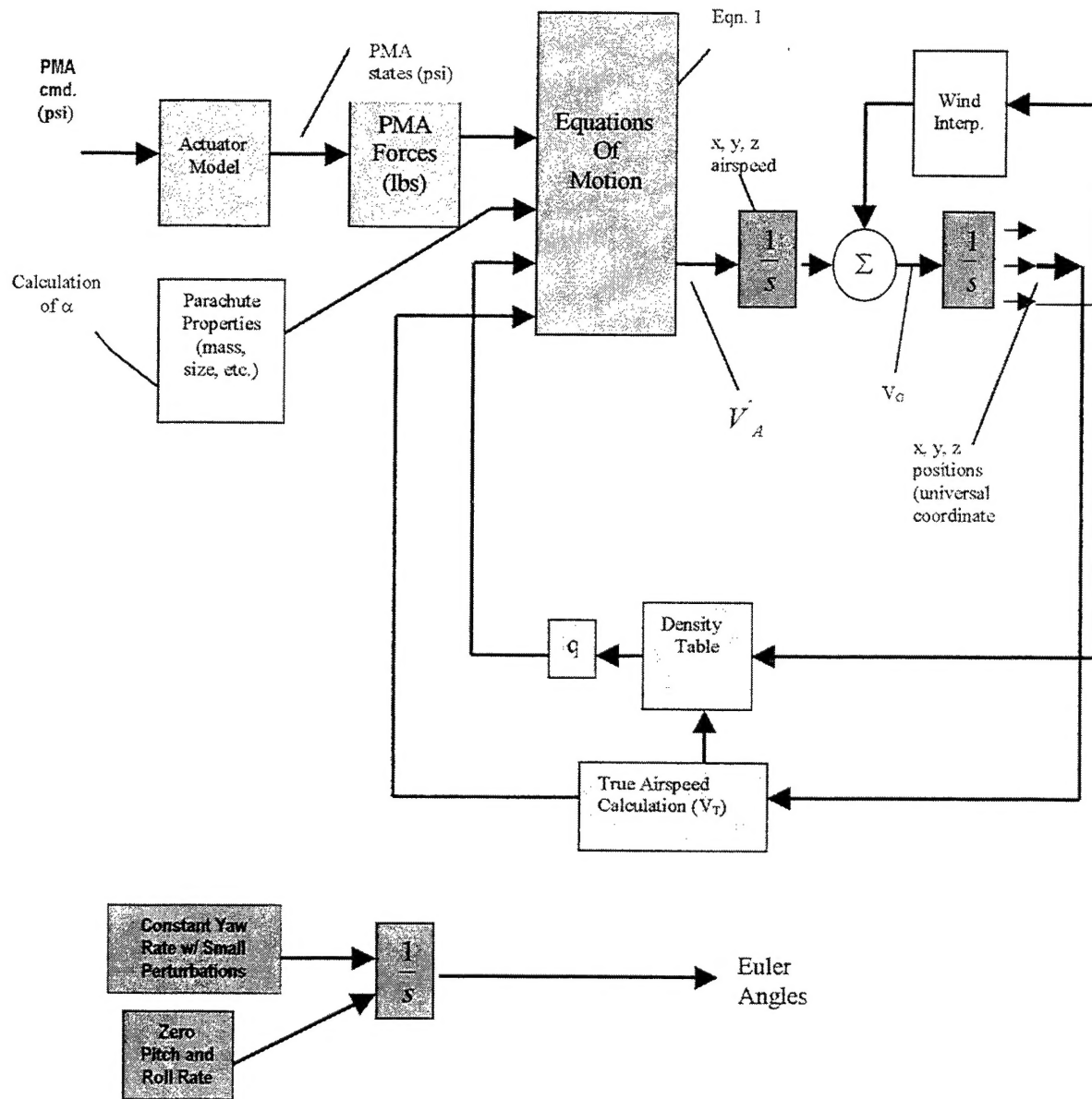
endfor;

delete CARP_x CARP_y predicted_x predicted_y predicted_z;

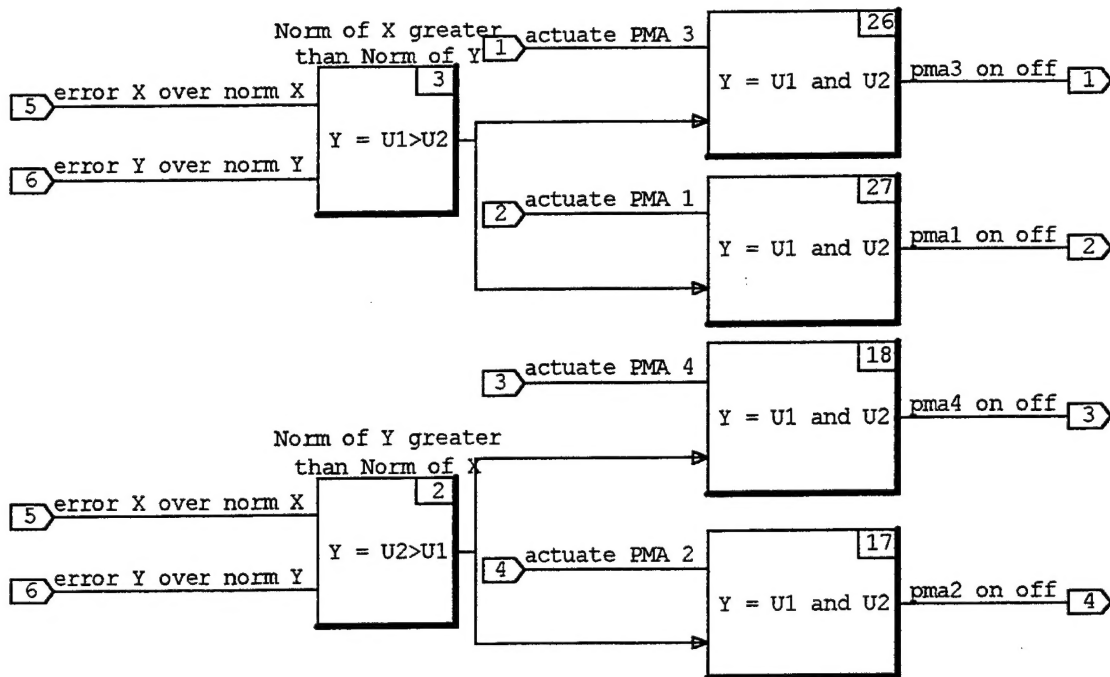
delete final_CARP_x final_CARP_y i actual_wind_choice forecast_wind_choice

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. ADDITIONAL FIGURE AND CODE



Simple Diagram of Parachute Point-mass Modeling Process



SystemBuild® Code for Allowing Only One Actuator at a Time

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
Naval Postgraduate School
411 Dyer Road
Monterey, CA 93943

3. Maximilian F. Platzler, Distinguished Professor.....1
Mail Code: AA/Pl
Dept. of Aeronautics and Astronautics
Monterey, CA 93943

4. Professor Biblarz.....1
Mail Code: AA/Bi
Dept. of Aeronautics and Astronautics
Monterey, CA 93943

5. Isaac I. Kaminer, Associate Professor.....1
Mail Code: AA/Ka
Dept. of Aeronautics and Astronautics
Monterey, CA 93943

6. Richard M. Howard, Associate Professor.....1
Mail Code: AA/Ho
Dept. of Aeronautics and Astronautics
Monterey, CA 93943

7. Oleg A. Yakimenko, Visiting Professor.....1
Mail Code: AA/Yk
Dept. of Aeronautics and Astronautics
Monterey, CA 93943

8. U.S. Army Soldier & Biological Chemical Command.....2
Soldier Systems Center Natick
ATTN: SSCNC-UTS (Richard Benney)
Kansas Street
Natick, MA 01760-5017

9. Technical Library.....1
US Army Yuma Proving Ground
STEYP-MT-A
Yuma, AZ 85365
10. US Army Yuma Proving Ground.....2
Aviation and Airdrop Systems Division
STEYP-MT-EA
Yuma, AZ 85365
11. ENS Timothy A. Williams.....1
Author
7201 Old Spanish Trail
Ocean Springs, MS 39564